GUIDE TO USING THE DISTRIBUTED CONTROL MODULES

Order Number: 146312-001

Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

CAUTION

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A Computing Device pursuant to Subpart J of Part 15 of FCC rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's software license, or as defined in ASPR 7-104.9(a)(9).

No part of this document may be copied or reproduced in any form or by any means without prior written consent of Intel Corporation.

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and its affiliates and may be used only to identify Intel products:

BITBUS	iLBX	iPDS	Plug-A-Bubble
COMMputer	im	iRMX	PROMPT
CREDIT	iMMX	iSBC	Promware
Data Pipeline	Insite	iSBX	QUEX
GENIUS	Intel	iSDM	QUEST
i	intel	iSXM	Ripplemode
i	intelBOS	Library Manager	RMX/80
I^2ICE	Intelevision	MCS	RUPI
ICE	inteligent Identifier	Megachassis	Seamless
iCS	inteligent Programming	MICROMAINFRAME	SOLO
iDBP	Intellec	MULTIBUS	SYSTEM 2000
iDIS	Intellink	MULTICHANNEL	UPI
	iOSP	MULTIMODULE	

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

*MULTIBUS is a patented Intel bus.

REV.	REVISION HISTORY	DATE
-001	Original Issue.	3/84

PREFACE

The Distributed Control Modules (DCM) product is a set of pre-packaged hardware and software products that you can use to construct high-speed, distributed, microcontroller systems. The DCM allows you to design an intelligent, low-cost network of microcontrollers which you can program for process control, environmental control, and other applications that require high-speed processing. The DCM product consists of the following packages:

- The DCM controller.
- The iSBX 344 BITBUS Controller Multimodule Board.
- The iRCB 44/10 Remote Controller Board.
- The iRMX 510 Support Package
- The iRMX 51 Executive (on diskette)

These packages and their functions are described in Chapter 2. Refer to the next section for information about how this manual is organized.

ORGANIZATION OF THIS MANUAL

This manual is divided into three major sections which contain the following information:

- **Introduction** — The introductory section of the manual explains the features of the DCM product. It briefly describes the modules which comprise the DCM system. In addition, it presents an example of a simplified DCM application.
- **Software Information** — The software section of the manual explains how to use the iRMX 51 Executive, the iRMX 510 Operating System handlers, and the other DCM software. This part of the manual is structured so that you can use it as a quick reference after you understand how the software works.

- Hardware Information — The hardware section of the manual describes how to use the DCM hardware to design a system of microcontrollers. It explains the design of Intel's implementation of a DCM controller-based module within a BITBUS environment. It then describes how to use the Intel-provided iSBX 344 MULTIMODULE board and the iRCB 44/10 board.

The following paragraphs describe the contents of each chapter of this manual.

INTRODUCTION

- Chapter 1 — This chapter describes the features of the DCM system and what it can do for you. It gives an overview of the system and introduces new terms.
- Chapter 2 — Chapter 2 describes the modules which make up the DCM system. It presents reasons for the packaging and explains each element within the module. These descriptions are high-level and thus refer to more specific chapters which describe the particular module part in detail.
- Chapter 3 — Chapter 3 is a sample DCM application. It is an introductory example which illustrates the features of a DCM system.

SOFTWARE INFORMATION

- Chapter 4 — Chapter 4 introduces the software included in the DCM system. It explains how the software is imbedded in the modules and discusses exactly where it resides.
- Chapter 5 — Chapter 5 introduces the iRMX 51 Executive. It explains the key concepts you need in order to understand and use this executive.
- Chapter 6 — Chapter 6 explains how to use the iRMX 51 services. It describes the services (system calls) in detail and gives an example of how to use each one.

- Chapter 7 — Chapter 7 describes how to configure the iRMX 51 Executive and how to configure the DCM controller firmware. You should refer to this chapter whether you are using the iRMX 51 Executive as a stand-alone system or you are using the DCM controller firmware as part of a larger DCM system.
- Chapter 8 — Chapter 8 describes the Remote Access and Control (RAC) interface. It describes the RAC commands in detail and explains how to use them.
- Chapter 9 — Chapter 9 describes how to use the iRMX 510 Operating System handlers to allow your DCM system to communicate with other Intel operating systems.

HARDWARE INFORMATION

- Chapter 10 — Chapter 10 introduces the hardware of the DCM product line. It discusses the relationship between the BITBUS Interconnect and the Distributed Control Modules.
- Chapter 11 — Chapter 11 describes the design of the fundamental BITBUS node on which all DCM boards are based. It also specifies the minimum design requirements for all BITBUS nodes.
- Chapter 12 — Chapter 12 explains how to configure the iSBX 344 BITBUS Controller MULTIMODULE board for use in BITBUS system. It relies heavily on Chapter 11 for details of board operation.
- Chapter 13 — Chapter 13 explains how to configure the iRCB 44/10 Remote Controller Board for use in BITBUS system. It relies heavily on Chapter 11 for details of board operation.
- Chapter 14 — Chapter 14 provides service information for the board products and contains the schematic diagrams for the iSBX 344 board and the iRCB 44/10 board.
- Appendix A — Appendix A lists the contents of some important DCM firmware files and some iRMX 51 files. These files include configuration files, external declaration files, libraries, and INCLUDE files.

READER LEVEL

This manual assumes that you are familiar with microcomputer programming concepts. You should also have a working knowledge of one of the following Intel programming languages:

- ASM51 Macro Assembly Language.
- PL/M-51 Programming Language.

In addition, you should be familiar with the BITBUS section of the DISTRIBUTED CONTROL MODULES DATA BOOK.

CONVENTIONS

This section describes the different conventions this manual uses in presenting information. The topics this section discusses are as follows:

- Conventions for section headings.
- Conventions for examples.
- Notational Conventions.

CONVENTIONS FOR SECTION HEADINGS WITHIN CHAPTERS

To facilitate an orderly presentation of information, this manual uses the following conventions for section headings within chapters:

- X.X FIRST-ORDER HEADING

  Chapters are divided into major sections. Each of these sections is called a first-order section, and each bears a first-order heading. First order headings appear in all "CAPS" with an underline.

- X.X.X SECOND-ORDER HEADING

  Each first-order section in a chapter can be divided into subsections. Each of these second-order sections is marked by a second-order heading. These headings are in all "CAPS" but without an underline.

- X.X.X.X Third-Order Heading

  Each second-order section can be divided into subsections called third-order sections. Each of these sections bears a third-order section. Third-order headings have initial "Caps" and no underline.

- X.X.X.X.X FOURTH-ORDER HEADINGS. Each third-order section can be divided into subsections called fourth-order sections. Each of these sections bears a fourth-order heading. These headings are in all "CAPS", are underlined, and the text begins on the same line as the section heading.

- X.X.X.X.X Fifth-Order Headings. Each fourth-order section can be divided into subsections called fifth-order sections. Each of these bears a fifth-order heading. These headings have initial "Caps", are underlined, and the text begins on the same line as the section heading.

CONVENTIONS FOR EXAMPLES IN THIS MANUAL

This manual contains a detailed example of a DCM application and a number of short programs which illustrate how to use the programming features of the DCM. These programs appear in both upper and lower case characters (for clarity); however, the DCM system does not distinguish between upper and lower case characters.

NOTATIONAL CONVENTIONS

This manual uses the following notations to describe the Distributed Control Modules:

Notation	Meaning
DCM	Distributed Control Modules
RAC	Remote Access and Control
SDLC	IBM Synchronous Data Link Control
RCB	Remote Controller Board

RELATED PUBLICATIONS

The following manuals and specifications provide additional information that may be helpful to you.

- PL/M-51 User's Guide, Order Number: 121966
- MCS-51 Macro Assembly User's Guide, Order Number: 980937
- Microcontroller Handbook, Order Number: 210918
- Microprocessor and Peripheral Handbook, Order Number: 210844
- Distributed Control Modules Data Book, Order Number: 230972-001
- Intel iSBX™ Bus Specification, Order Number: 142686-002
- iPDS™ Personal Development System User's Guide, Order Number: 162606
- iRMX™ 86 Nucleus Reference Manual, Order Number: 9803122
- iRMX™ 88 Reference Manual, Order Number: 142323
- EIA RS485 Electrical Standard Proposal, (April 6, 1983), Number 1488

CONTENTS

CONTENTS (continued)

CONTENTS (continued)

TABLES

FIGURES

CHAPTER 1
WHAT IS THE DCM?

The DCM product is a distributed system of hierarchical, microcontroller nodes and complimenting software. The nodes communicate through a high-speed serial bus (the BITBUS interconnect) which allows them to exchange information.

This chapter is a high-level introduction which describes the DCM product and what it can do for you. Later chapters explain the details of the system.

1.1 INTRODUCTION TO THE DCM SYSTEM

The DCM system consists of a master node, up to 250 slave nodes, and preconfigured software that you can use to control processes within each node. The main purpose of the DCM system is to control dedicated functions. (Each node can independently control functions.) The DCM system is an interrupt-driven system and thus it responds to events which occur in the outside world.

The DCM system provides solutions to the problems which are usually associated with microcontroller systems. The following list describes some common problems area in microcontroller systems and explains how the DCM product provides solutions for these problems areas:

- **Computing speed between the master nodes and the slave nodes.** -- Most microcontroller systems are designed so that the master node makes all of the decisions. The DCM system eliminates the "bottleneck" at the master node by providing event-controlling feedback at the local node.

  DCM slave nodes can contain intelligence and can, therefore, make decisions at the local level without accessing the master node. However, there are times when the slave must access the master and the DCM system provides a very high-speed serial communications bus for those times.

- **Controlling events which are physically separate.** -- Microcontroller systems typically have problems dealing with physically separate events. Excessive cabling and costly master devices are usually characteristics of this type of system. The DCM system is designed to control physically separate events. Each slave node can contain intelligence (firmware) which permits the DCM system to process events concurrently at each node. The DCM system also contains a high-speed, serial bus (mentioned previously) which connects the nodes so that they can exchange information with the master node while executing concurrently.

Figure 1-1 illustrates the concepts just discussed; the master node instructs the slave node to double the pulse rate of a remote clock. The slave node changes the rate and sends the new pulse rate back to the master.

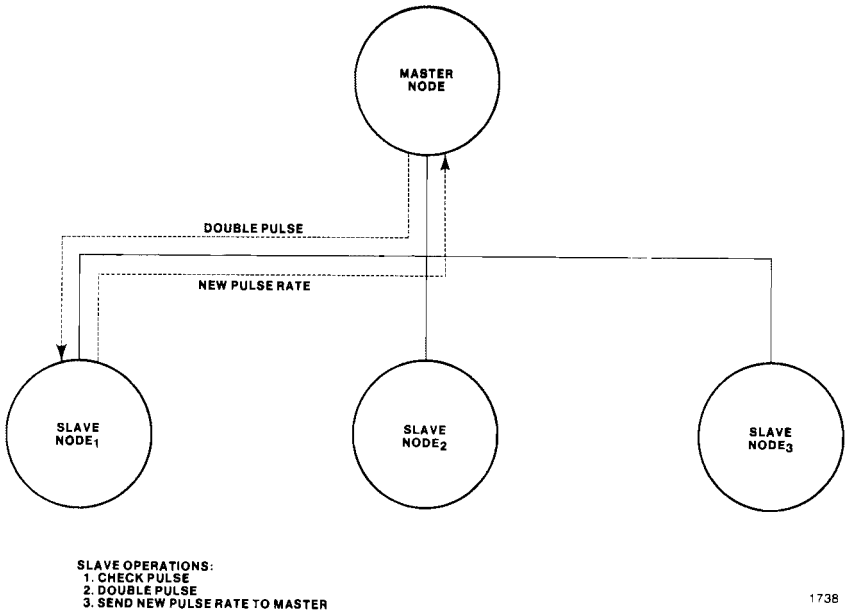


Figure 1-1. Introductory Concepts for the DCM System

The rest of this chapter describes the features of the DCM product which allow you to avoid the problems described in this section.

1.2 FEATURES OF THE DCM SYSTEM

This section describes the hardware and software features which make the DCM system unique among distributed microcontroller systems. The following subsections briefly explain the functions of the parts included in the DCM system.

Refer to the Chapter 2 for information regarding the particular packaging of the modules. Chapter 2 describes the individual parts in detail and explains the form in which they are available.

1.2.1 MASTER AND SLAVE NODES

The DCM system contains two types of boards that you can use as either master or slave nodes: the iRCB 44/10 Remote Controller board and the iSBX 344 BITBUS Controller MULTIMODULE board. The iRCB 44/10 Remote Controller Board is a small form factor (single-wide Eurocard) single board computer that provides a low-cost BITBUS node with I/O capability.

The iRCB 44/10 board is based on the DCM controller, a firmware-enhanced 8044 microcontroller. The remote controller board features 24 parallel I/O lines, with which you can monitor and modify outside events, and an iSBX connector for additional I/O expansion. The board also includes sockets for a BITBUS repeater and on-board ROM and RAM sockets for the DCM controller's external memory needs and application programs. You can create a simple DCM system by connecting these boards through the BITBUS interface as shown in Figure 1-2. Refer to the BITBUS section of the DISTRIBUTED CONTROL MODULES DATA BOOK for more information.

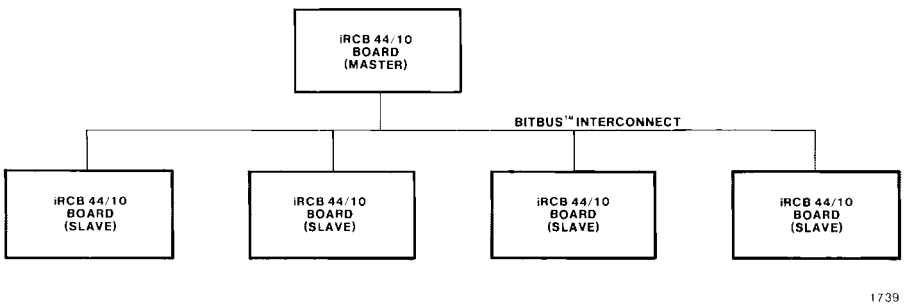


Figure 1-2. Simple DCM System of iRCB 44/10 Boards Connected by the BITBUS™ Interface

The iSBX 344 BITBUS Controller Multimodule board is double-wide iSBX Multimodule based on the DCM controller. The iSBX 344 board has a BITBUS connection, plus a parallel iSBX connection. The BITBUS connection allows the iSBX 344 board to be part of a DCM system, and the parallel iSBX connection allows the iSBX 344 board to communicate with any board that supports the iSBX I/O Expansion Bus. The iSBX connector enables you to add master or slave extensions to your DCM system, and allows these extensions to have processors with different capabilities than the 8044 microcontroller. The iSBX 344 board also features on-board ROM and RAM sockets for the DCM controller's external memory and application programs.

Figures 1-3 and 1-4 show two ways in which you could use the iSBX 344 board in a DCM system. The system in Figure 1-3 uses the iSBX 344 board and an Intel iSBC board as slave extension in a DCM system. This connection allows the iSBC processor board to communicate with the DCM system.

The system in Figure 1-4 uses the iSBX 344 board and an Intel iSBC board as a master extension in a DCM system. This connection gives the master node more capabilities and it allows the iSBC processor board to communicate with the DCM system.

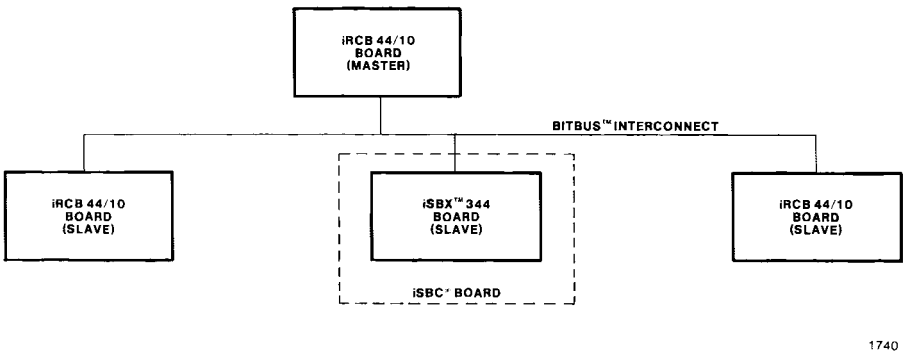


Figure 1-3. DCM System with Slave Extension

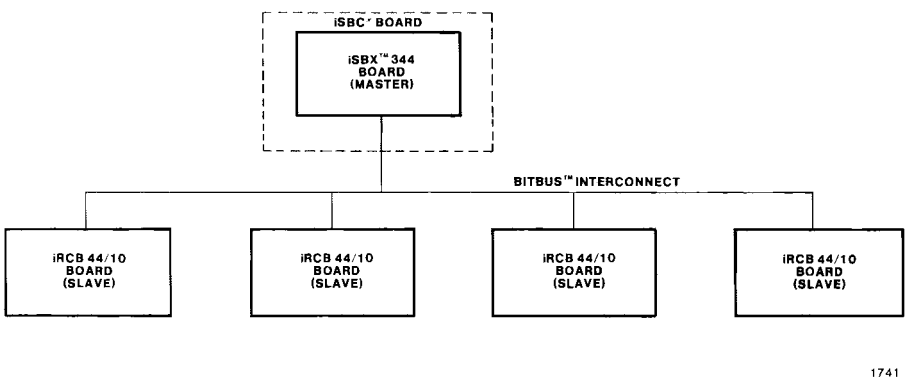


Figure 1-4. DCM System with Master Extension

1.2.2 COMMUNICATING WITH OTHER INTEL OPERATING SYSTEMS

The DCM system can also communicate with Intel operating systems which run on top of the processor boards to which the iSBX 344 board is attached. The iRMX 510 Operating System handler (of the DCM system) allows you to perform this function. You can use your operating system as either part of a slave node or part of the master node on the DCM system.

Figures 1-5 and 1-6 illustrate using the processor's operating system as part of a slave node and as part of the master node, respectively.

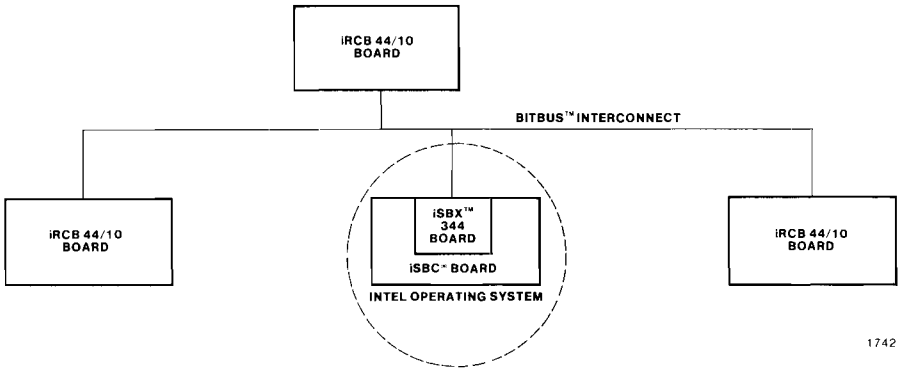


Figure 1-5. Using an Intel Operating System as a Slave Extension

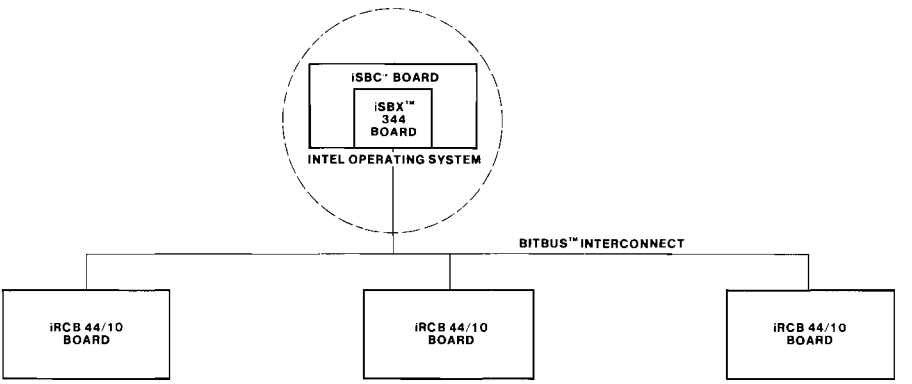


Figure 1-6. Using an Intel Operating System as a Master Node Extension

1.2.3 PASSING MESSAGES BETWEEN NODES

The DCM system provides you with an executive which allows you to send messages between nodes. The iRMX 51 Executive provides you with "system calls" which allow you to create tasks (programs) that reside on individual nodes. These tasks can then send simple or complex messages to one another over the BITBUS interconnect.

Chapter 5 explains system calls, tasks, and sending messages in more detail. Figure 1-7 illustrates the concept of message passing between nodes over the BITBUS interconnect. In this example, the master polls the task on Slave $Node_1$; the slave then responds by sending a message back to the master node. The master node then sends a message to the task on Slave $Node_3$.

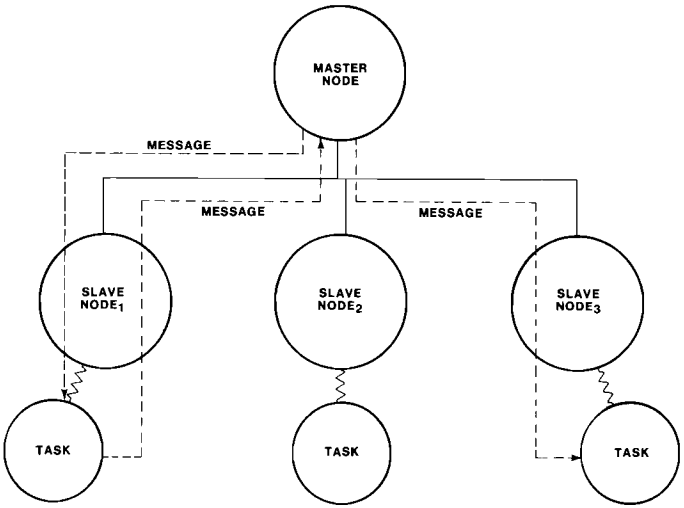


Figure 1-7. Message Passing Between Nodes

1.2.4 MULTITASKING INTELLIGENCE ON NODES

In addition to providing message passing capabilities between single tasks, the iRMX 51 Executive also provides multitasking capabilities. Multitasking means that a DCM system can have multiple tasks residing on the same node. Because the DCM system consists of intelligent nodes, each node (that contains the iRMX 51 Executive) can have a number of tasks that share the processor and execute one at a time.

Figure 1-8 illustrates the concepts of multitasking and message passing. In this example, the master node polls $Task_2$ on Slave $Node_1$; $Task_2$ responds by sending a message back to the master node. The master node then sends a message to $Task_1$ on Slave $Node_3$. Refer to Chapter 5 for more information about multitasking.

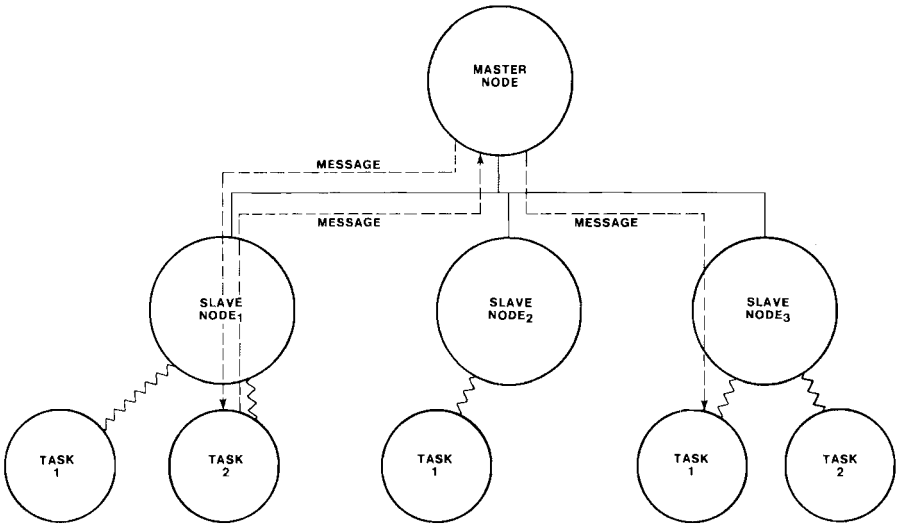


Figure 1-8. Multitasking on a DCM System

1.2.5 HANDLING INTERRUPTS FROM THE OUTSIDE WORLD

The iRMX 51 Executive allows the nodes on your DCM system to accept, prioritize, and interpret events occurring in the outside world. This means that remote nodes can act upon events when they occur without, necessarily, accessing the master node. Refer to Chapter 5 for more information about how the iRMX 51 Executive handles interrupts.

***

CHAPTER 2
HOW THE DCM PRODUCT IS PACKAGED AND WHY

This chapter describes the modules included in the DCM system. It lists reasons for the specific packaging and explains each element of the particular module. The sections in this chapter explain the modules in a high-level manner to help you understand the DCM system as a whole; therefore, the sections contain references to other chapters which contain more specific information about each part of the module.

2.1 INTRODUCTION TO THE PACKAGING OF THE DCM MODULES

The modules within the DCM system are packaged so that you can use them in a number of different design situations. The modules are available in the following forms:

- A group of preconfigured components, boards, and software that you can use to form a DCM system.
- Individual boards.
- Individual components.

Intel has engineered this system so that you can use the modules as a system-level, preconfigured, distributed system of microcontrollers, as a board-level product to incorporate into a custom system, or as a component-level product to include on a custom-designed board.

2.1.1 USING THE DCM PRODUCT AS A PRECONFIGURED, DISTRIBUTED SYSTEM OF MICROCONTROLLERS

You can use the parts of the DCM product to form a distributed system of microcontrollers. Intel provides the DCM product as a collection of boards connected by the BITBUS interconnect for customers who need a microcontroller system to manage a set of processes. These customers can use the DCM product as a system rather than designing their own microcontroller system. By using the DCM product as a system, you can control your process without investing time and money designing your own system.

Instead, you can spend your investment dollars perfecting the way in which the DCM product monitors and/or controls the processes.

2.1.2 USING THE DCM PRODUCT IN BOARD-LEVEL PRODUCTS

Intel provides the DCM system in a series of board-level modules for customers who wish to incorporate DCM products in their custom-designed microcontroller systems. This kind of packaging allows customers who design special-purpose systems to take advantage of the DCM controller and board capabilities. This reduces your board development time and thus the final cost of your product.

2.1.3 USING THE DCM PRODUCT IN COMPONENT-LEVEL PRODUCTS

Intel also provides parts of the DCM product in component modules for customers who wish to design their own special-purpose microcontroller boards. This kind of packaging allows customers who design specific-purpose systems to take advantage of the DCM controller capabilities without including DCM features they may not require.

2.2 SPECIFIC PACKAGING OF THE MODULES

This section explains the specific way in which Intel packages the DCM system. You can purchase these modules separately or in a system as explained in the previous section. This section explains the contents of the following four modules:

- The DCM Controller.
- The iSBX 344 BITBUS Controller MULTIMODULE Board.
- The iRCB 44/10 Remote Controller Board.
- The iRMX 510 Support Package.
- The iRMX 51 Executive.

2.2.1 THE DCM CONTROLLER

The DCM controller is an Intel 8044 component with the DCM controller firmware in on-chip Read Only Memory (ROM). Figure 2-1 illustrates the parts which make up the DCM controller.

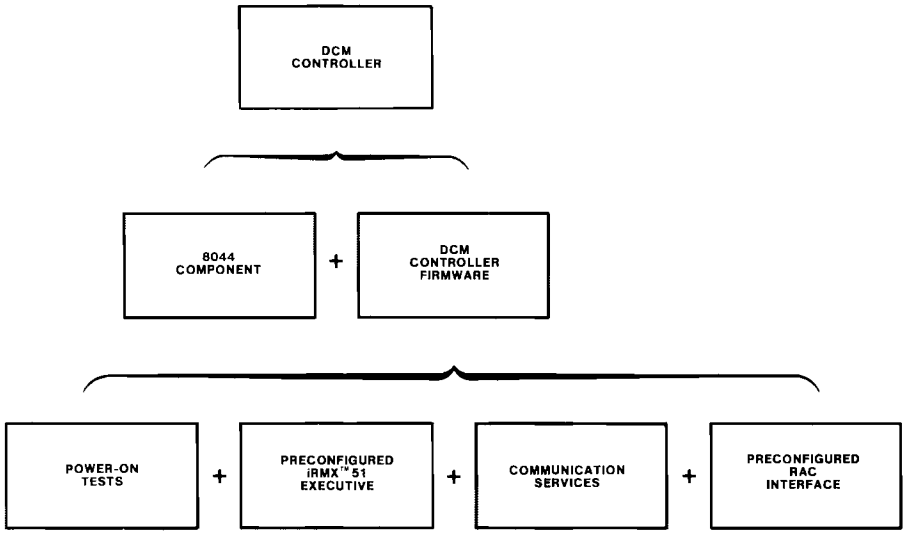


Figure 2-1. Parts Which Make Up the DCM Controller

The following sub-sections briefly describe the parts which make up the DCM controller.

2.2.1.1 The 8044 Microcontroller Component

The 8044 component is an intelligent, programmable microcontroller. The DCM controller incorporates the DCM controller firmware into the 8044 component. Refer to Intel's MICROCONTROLLER HANDBOOK for more information about the 8044 component. Refer to the following section for more information about the DCM controller firmware.

2.2.1.2 The DCM Controller Firmware

The DCM controller firmware consists of the following parts:

- Power-on tests.
- Preconfigured iRMX 51 Executive.
- Communications services.
- Preconfigured RAC interface

The DCM controller firmware resides on the 8044 component in ROM. The firmware is preconfigured so that you do not have to modify anything in order to use the DCM controller. The following sub-sections describe the parts which make up the DCM controller firmware.

2.2.1.2.1 POWER-ON TESTS. The power-on and diagnostics tests are part of the DCM controller firmware which resides on the DCM controller. The tests automatically check your system for hardware problems. These checks consist of four tests to determine if your hardware is functioning correctly. Refer to Chapter 11 in the Hardware section for more information about the tests and the functions they check.

2.2.1.2.2 PRECONFIGURED iRMX™ 51 EXECUTIVE. The iRMX 51 Executive is part of the DCM controller firmware which resides on the DCM controller. The iRMX 51 Executive is an executive for the 8051/8052 microcontrollers. It is a real-time executive that allows you to send messages between both tasks local to the processor and tasks residing on other processors.

The concept of a real-time executive, local and remote tasks, message passing, and other iRMX 51 system-related topics is explained in Chapter 5. Chapter 6 explains how to use specific iRMX 51 services (system calls).

2.2.1.2.3 COMMUNICATIONS SERVICES. The DCM firmware also includes communication services. These services are available through the BITBUS interface and the parallel interface. The interfaces communicate with the boards through a network of gateways.

The DCM controller firmware handles the communication services automatically. Refer to Chapter 13 for more information about the parallel interface. Refer to the BITBUS section of the DISTRIBUTED CONTROL MODULES DATA BOOK for more information about the BITBUS interface.

2.2.1.2.4 PRECONFIGURED RAC INTERFACE. The Remote Access and Control (RAC) interface is also part of the DCM controller firmware which resides on the DCM controller. The RAC interface is part of a preconfigured iRMX 51 task that resides on all DCM nodes. It can perform a specific set of services which are described in Chapter 8.

2.2.2 THE iSBX™ 344 BITBUS™ CONTROLLER MULTIMODULE™ BOARD.

The iSBX 344 BITBUS Controller MULTIMODULE board contains a DCM controller plus a BITBUS connection and a parallel iSBX connection.

Because the iSBX 344 board contains both a BITBUS connection and an iSBX connection, you can use it to allow other Intel boards to communicate with a DCM system. Figure 2-2 illustrates the (conceptual) parts which make up the iSBX 344 MULTIMODULE board.

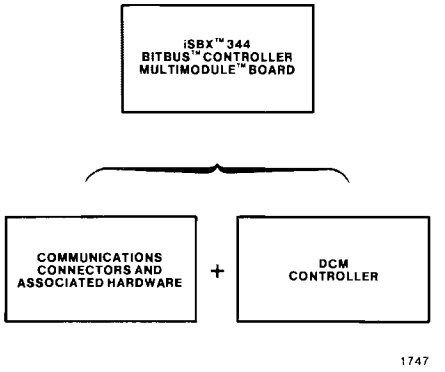


Figure 2-2. Parts of the iSBX™ 344 MULTIMODULE™ Board

The following sub-sections briefly describe the parts which make up the iSBX 344 MULTIMODULE board.

2.2.2.1 DCM Controller

The iSBX 344 board contains a DCM controller as its main processor. The parts of the DCM controller are explained in the previous section. If you use an Intel iSBC processor board with the iSBX 344 board, you can not only off-load user programs to the extension processor, but you can have user programs existing on both boards.

2.2.2.2 Communication Connections and Associated Hardware

The iSBX 344 MULTIMODULE board contains both an iSBX connector and a BITBUS connector. The iSBX connector allows you to attach the iSBX 344 board to other Intel boards; the iSBX 344 board can then communicate through both the parallel connection (to the iSBC board) and the BITBUS connection.

2.2.3 iRCB 44/10 REMOTE CONTROLLER BOARD

The iRCB 44/10 board is a single-wide eurocard board which has a BITBUS connection, an iSBX I/O expansion connection, the DCM controller and I/O hardware. The I/O hardware of the iRCB 44/10 board allows the DCM system to monitor and control events in the outside world. Figure 2-3 illustrates the (conceptual) parts which make up the iRCB 44/10 board.

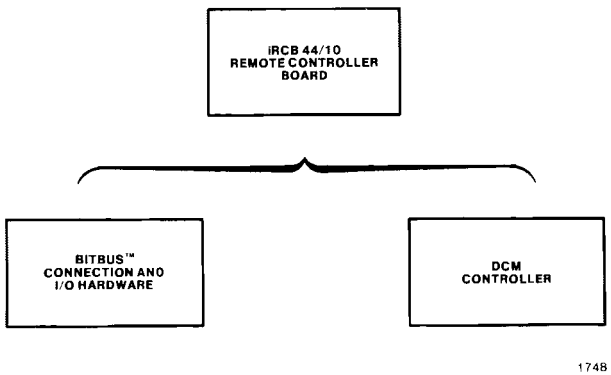


Figure 2-3. Parts of the iRCB 44/10 Board

2.2.3.1 The BITBUS™ Connection and the I/O Hardware

The BITBUS connection allows you to connect the iRCB 44/10 into a DCM system of BITBUS nodes. The I/O hardware allows the iRCB 44/10 (and thus the DCM system) to monitor and control external events.

2.2.3.2 The DCM Controller

The iRCB 44/10 board contains a DCM controller as its main processor. The DCM controller allows the iRCB 44/10 board to make monitoring/controlling decisions based on the information it receives through the I/O hardware. The parts of the DCM controller are explained in a previous section in this chapter.

2.2.4 THE iRMX™ 510 SUPPORT PACKAGE

The iRMX 510 Support Package resides on a flexible diskette; it consists of the following parts:

- DCM controller firmware.
- iRMX 510 Operating System handlers.
- DCM controller INCLUDE files and the iRMX 51 interface libraries.

You can use these parts to communicate with other Intel operating systems. Figure 2-4 illustrates the parts of the iRMX 510 Support Package.

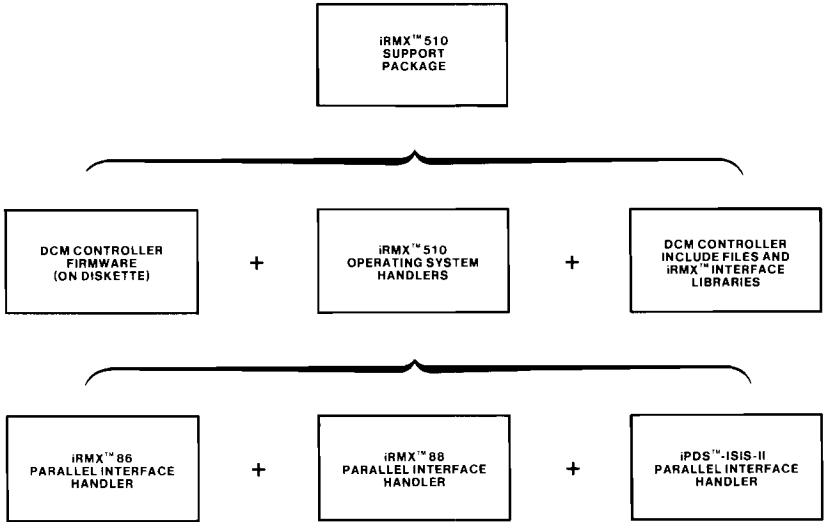


Figure 2-4. Parts of the iRMX™ 510 Support Package

The following sections describe the parts of the iRMX 510 Support Package in more detail.

2.2.4.1 The DCM Controller Firmware

The DCM controller firmware that resides on the diskette is the same as the DCM controller firmware on the DCM controller. Intel provides the DCM controller firmware in this form for use with the ICE-44 in-circuit emulator. Refer to previous sections for the information on the specific parts of the DCM controller firmware.

2.2.4.2 iRMX™ 510 Operating System Handlers

The iRMX 510 Operating System handlers allow you to communicate through the iSBX connection using the following Intel systems:

- iRMX 86 and iRMX 286R Operating Systems.
- iRMX 88 Executive.
- iPDS/ISIS-II Operating System on the iPDS Personal Development System.

The following subsections describe the parts of the iRMX 510 handlers.

2.2.4.2.1 iRMX™ 86 PARALLEL INTERFACE HANDLER. The iRMX 86 parallel interface handler allows the iRMX 86 and iRMX 286 Operating Systems to communicate with DCM systems. Chapter 9 explains the iRMX 86 parallel interface handler in detail.

2.2.4.2.2 iRMX™ 88 PARALLEL INTERFACE HANDLER. The iRMX 88 parallel interface handler allows the iRMX 88 executive to communicate with DCM systems. Chapter 9 explains the iRMX 88 parallel interface handler in detail.

2.2.4.2.3 iPDS™ ISIS-II PARALLEL INTERFACE HANDLER. The iPDS-ISIS-II parallel interface handler allows the iPDS operating system (ISIS-II) to communicate with DCM systems. Chapter 9 explains the iPDS/ISIS-II parallel interface handler in detail.

2.2.4.3 DCM CONTROLLER INCLUDE Files and iRMX™ 51 Interface Libraries

The iRMX 510 Support Package contains some INCLUDE files and the iRMX 51 interface libraries. These files can assist you in writing iRMX 51 applications.

Refer to Chapters 7 and 9 for more information about the DCM controller INCLUDE files and the iRMX 51 interface libraries; Appendix A lists the contents of these files.

2.2.5 THE iRMX™ 51 EXECUTIVE

In addition to residing on the DCM controller as a preconfigured executive, the iRMX 51 Executive is also available on a flexible diskette. In this form, the iRMX 51 Executive is a configurable, real-time executive for the 8051/8052 microcontrollers. The iRMX 51 diskette contains everything you need to set up a stand-alone executive for the 8051/8052 microcontrollers.

The concept of a real-time executive, tasks, message passing, and other iRMX 51 system-related topics is explained in Chapter 5. Chapter 6 explains how to use specific iRMX 51 services (system calls). Chapter 7 describes how to configure a stand-alone iRMX 51 System.

***

CHAPTER 3
A SAMPLE DCM APPLICATION

This chapter contains an example which illustrates some key features of a DCM system. It is intended only as an example of how you can use the elements of the DCM product to form a system that controls a process; therefore, the configuration and implementation details are not explained. You must refer to specific chapters in this manual for more detailed (but generic) information.

3.1 THE SITUATION

The object of this example is to control the speed and direction of a stepper motor from either a node with a console, or a node with a joystick. A remote alphanumeric display must also record the speed and direction of the motor or allow ASCII characters from the console to be passed to it.

One node must be the master controller (either the "console" node, or the "joystick" node). But, the other node must be able to control the motor, if necessary.

3.2 THE SOLUTION

This section offers a solution to the situation described previously. The following sections describe the solution both in a conceptual manner and in a physical manner. This method of explanation should help you to understand how the DCM system can apply to other applications.

3.2.1 THE CONCEPTUAL SOLUTION

Figure 3-1 is one (conceptual) solution to this control problem. $Node_1$ controls the speed and direction of the motor from console input. $Node_2$ (a slave) can control the speed and direction of the motor from a joystick input. $Node_3$ monitors the speed and direction of the motor on a remote alphanumeric display; it also receives ASCII characters from the master node. $Node_4$ actually operates the stepper motor from the directions of either $Node_1$ or $Node_2$.

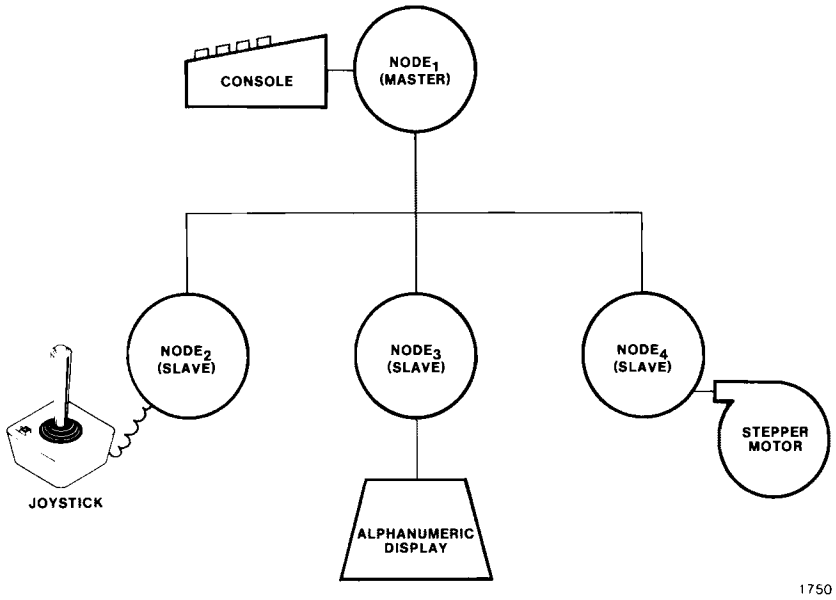


Figure 3-1. Conceptual System to Control a Stepper Motor

3.2.2 THE PHYSICAL SOLUTION

Now that we have a conceptual DCM system, we must come up with actual nodes that perform the services we require. Figure 3-2 is one physical solution to this control problem.

We've substituted an iSBC 344 board plus an extension (System 86/330) for $Node_1$ and an iRCB 44/10 board for $Node_2$, $Node_3$, and $Node_4$. (Three iRCB 44/10 boards are not required for this application; this example uses three boards simply for illustration purposes.) The next section describes the parts of each node and the services they perform.

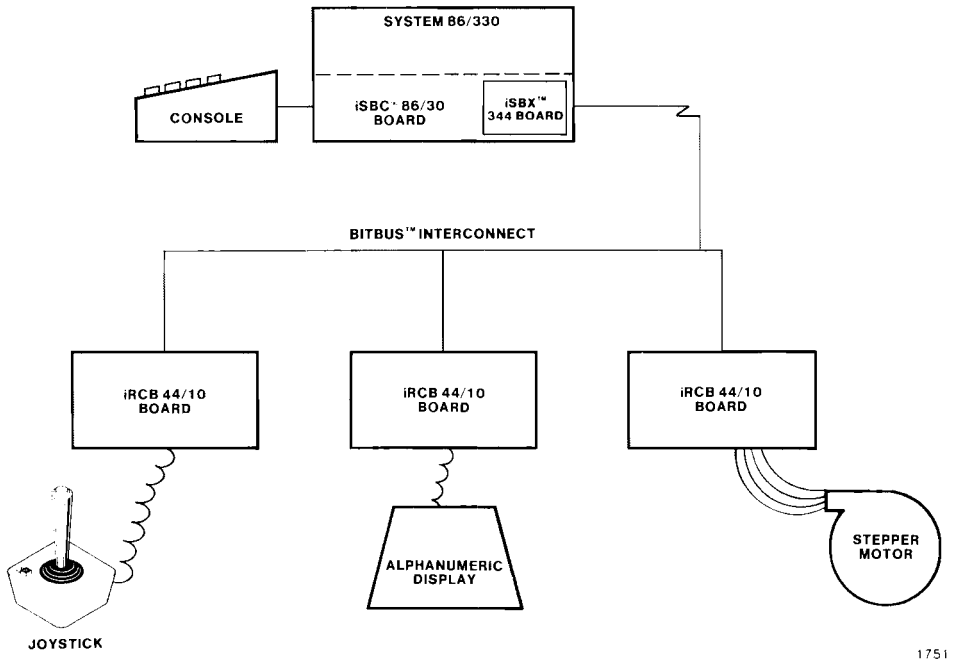


Figure 3-2. Physical System to Control a Stepper Motor

3.2.2.1 $Node_1$ -- The Master Node

This solution uses the iSBX 344 BITBUS Controller MULTIMODULE board connected to an iSBC 86/30 board by the iSBX connector as the master node ($Node_1$). The iSBC 86/30 board is part of a System 86/330 which includes the following software:

- A complete iRMX 86 Operating System, the iRMX 510 Operating System handler, and the iRMX 51 Executive (DCM firmware) on the iSBX 344 board.
- A terminal handling procedure.
- A message handling procedure.

By using the System 86/330 as a master extension, it can poll all nodes in succession. The terminal handling procedure produces the following menu on the System 86/330:

1) Start
2) Stop
3) Speed
4) Display

"Start" initializes the system and instructs it to begin polling the nodes (via the BITBUS interconnect). "Stop" halts all communication between the master and slave nodes. "Speed" instructs the system to disregard the console and to direct the motor from the joystick on $Node_2$. "Display" either instructs the alphanumeric display to record (analog display of) the joystick-directed motor speed and direction or allows characters typed at the console to override the joystick information..

3.2.2.2 $Node_2$ -- The Joystick

$Node_2$ consists of an iRCB 44/10 Remote Controller Board and a joystick. The joystick drives the input lines on the iRCB 44/10 board and uses the RAC task to receive the port number from the master node.

3.2.2.3 $Node_3$ -- Alphanumeric Display

$Node_3$ consists of an iRCB 44/10 Remote Controller Board and an alphanumeric display. This node contains some user software that allows it to wait for a message, receive ASCII characters and echo them to the display, and detect when the master is no longer polling it.

3.2.2.4 $Node_4$ -- Stepper Motor

$Node_4$ simply takes instructions from $Node_1$; $Node_1$ can give $Node_4$ instructions directly, or it can poll $Node_2$ and relay those directions.

***

CHAPTER 4
SOFTWARE INCLUDED IN THE DCM PRODUCT

This chapter discusses the software included in the DCM system. The software is imbedded in the modules described in Chapter 2. This chapter explains exactly where (and in what forms) the software exists; it also points to other sections of this manual where you can find more detailed information.

The DCM software resides in the following modules:

- The DCM controller firmware.
- The iRMX 510 Support Package.
- The iRMX 51 Executive.

You can use the software within these modules in any combination. You can include the software that is useful in your application and you can omit the software that does not pertain to your particular application. For example, if you do not wish to communicate with another operating system you can omit the iRMX 510 Support Package software.

The following sections introduce you to the software within the modules and help you decide the software your applications require.

4.1 THE DCM CONTROLLER FIRMWARE

The DCM controller firmware resides on the DCM controller; it consists of the following four parts:

- Power-up and diagnostics tests.
- iRMX 51 Executive in preconfigured firmware.
- Communications services (BITBUS and parallel bus interfaces).
- Preconfigured Remote Access and Control (RAC) interface.

This section describes these parts and their functions. It then refers you to chapters of the manual which describe the parts in more detail.

4.1.1 POWER-UP AND DIAGNOSTICS TESTS

When you power-on or reset your DCM system, the power-up and diagnostics tests automatically check the DCM controller hardware functions. These checks consist of four tests to determine if your hardware is functioning correctly. Refer to Chapter 11 in the Hardware section for more information about the tests and the functions they check.

4.1.2 THE iRMX™ 51 EXECUTIVE IN PRECONFIGURED FIRMWARE

The iRMX 51 Executive is an executive for the 8051 family of microcontrollers. It is a real-time executive that allows you to send messages between both local and remote tasks (using iRMX 51 Executive and the RAC interface). Chapter 5 explains the concept of a real-time executive, local and remote tasks, message passing, and other iRMX 51 system-related topics. Chapter 6 explains how to use specific iRMX 51 services (system calls).

A preconfigured iRMX 51 Executive is part of the DCM controller firmware which resides on the DCM controller. (The iRMX 51 Executive is also available on a flexible diskette for applications other than DCM systems; see a later section in this chapter.)

4.1.3 COMMUNICATIONS SERVICES

The DCM firmware also includes communication services in the form of a BITBUS interface and a parallel interface. These interfaces provide communication with the boards through a network of gateways.

The DCM firmware automatically handles the communication services. Refer to Chapter 13 for more information about the parallel interface. Refer to the BITBUS section of the DISTRIBUTED CONTROL MODULES DATA BOOK for more information about the BITBUS interface.

4.1.4 PRECONFIGURED RAC INTERFACE

The Remote Access and Control (RAC) interface is a preconfigured task that is supplied on every DCM controller. The RAC task provides a means to manipulate memory and I/O areas. By using the RAC task, you can manage tasks on remote nodes without (necessarily) creating a task. Chapter 8 contains detailed descriptions of the individual RAC services that perform these functions.

4.2 iRMX™ 510 SUPPORT PACKAGE

The iRMX 510 Support Package is shipped on flexible diskettes; it consists of the following three parts:

- iRMX 510 Operating System handlers.
- DCM controller firmware (as described in the previous section, but residing on a flexible diskette).
- DCM controller INCLUDE files and the iRMX 51 interface libraries.

The type of diskette you use depends upon the kind of machine you are using to load the files. The diskettes contain the same files, however, they are in different formats. The formats are as follows:

- Single-sided, single-density, ISIS-II 8-inch format.
- Single-sided, double-density, ISIS-II 8-inch format.
- Double-sided, double-density, iRMX 86 8-inch format.
- Double-sided, double-density, iPDS 5 1/4-inch diskette.

The next three sections describe the iRMX 51 Support Package parts and their functions. Each section then refers you to chapters of the manual which describe the parts in more detail.

4.2.1 iRMX™ 510 OPERATING SYSTEM HANDLERS

The iRMX 510 Operating System handlers provide you with a way of communicating with Intel Operating Systems. This software interface allows you to access the iRMX 51 services through the following Intel operating systems:

- iRMX 86 Operating System.
- iRMX 88 Executive.
- iRMX 286R Operating System.
- iPDS/ISIS-II Operating System on the iPDS Personal Development System.

In order to use the iRMX 510 Support Package you must plug an iSBX 344 BITBUS Controller MULTIMODULE board into the iSBX connection on your processor board. Refer to Chapter 9 for more information about the iRMX 510 Operating System handlers and the operating systems they allow you to access. Refer to Chapter 13 for more information about the iSBX 344 MULTIMODULE board.

4.2.2 DCM CONTROLLER FIRMWARE (IN A LOADABLE OBJECT FILE) ON A FLEXIBLE DISKETTE

The DCM controller firmware on this diskette contains all the parts mentioned in the previous section: the power-up and diagnostics tests, the iRMX 51 Executive, communications services (BITBUS and parallel bus interface), and the RAC interface.

However, because the DCM controller firmware is in a loadable object file on the flexible diskette, you can download the contents of the file from another Intel system (through the ICE-44 In-Circuit Emulator).

Chapter 7 of this manual describes how to configure the iRMX 51 Executive to meet your particular needs.

Refer to Chapter 11 in the Hardware section for more information about the power-up and diagnostics tests and the functions they check. Refer to Chapters 5 and 6 for more information about the iRMX 51 Executive. Refer to Chapter 12 for more information about the parallel interface. Refer to the BITBUS section of the DISTRIBUTED CONTROL MODULES DATA BOOK for more information about the BITBUS interface. Refer to Chapter 8 for detailed descriptions of the RAC commands.

4.2.3 DCM CONTROLLER INCLUDE FILES AND THE iRMX™ 51 INTERFACE LIBRARIES

The DCM firmware contains the INCLUDE files and the iRMX 51 interface libraries that resolve iRMX 51 externals and include data types. When you use the version of the DCM firmware on the DCM controller, these files and libraries are automatically placed in the correct area.

If you are using the DCM controller firmware in its software (loadable object file) form, you may want to use the INCLUDE files and the interface libraries when writing additional iRMX 51 user tasks for your DCM system. Appendix A lists the contents of the DCM INCLUDE files; you can change the literal declarations in these files if necessary. However, you should not modify the iRMX 51 interface libraries.

4.3 THE iRMX™ 51 EXECUTIVE

The iRMX 51 Executive is also available as a configurable, real-time executive for the 8051/8052 microcontrollers. This form of the executive is shipped on a flexible diskette.

The type of diskette you use depends upon the kind of machine you are using to load the executive. The diskettes contain the same files, however, they are in different formats. The formats are as follows:

- Single-sided, single-density, ISIS-II 8-inch format.
- Single-sided, double-density, ISIS-II 8-inch format.
- Double-sided, double-density, iPDS 5 1/4-inch diskette.

The diskette contains everything you need to set up a stand-alone executive for the 8051/8052 microcontrollers. The concept of a real-time executive, tasks, message passing, and other iRMX 51 system-related topics is explained in Chapter 5. Chapter 6 explains how to use specific iRMX 51 services (system calls). Chapter 7 describes how to configure a stand-alone iRMX 51 System.

***

CHAPTER 5
INTRODUCTION TO THE iRMX™ 51 EXECUTIVE

This chapter explains the purpose of an executive and, specifically, the features of the iRMX 51 Executive. It describes iRMX 51 terminology and architecture so that you can use the system more effectively.

The iRMX 51 Executive is available as part of the DCM system and as a stand-alone executive for the 8051/8052 microcontrollers.

5.1 WHAT IS AN EXECUTIVE AND WHY SHOULD I USE ONE?

An executive is a shortcut to the marketplace. It supplies you with features that most applications require, thus allowing you to focus your attention on developing your application software. When you use an executive, you spend less time and effort developing system software; therefore, you can bring your application to the market faster and at a lower price.

The next two sections describe different types of executives that relate to the iRMX 51 Executive.

5.1.1 REAL-TIME EXECUTIVES

Real-time executives monitor events in the outside world. These events occur at seemingly random intervals for which a system cannot plan. Therefore, real-time executives are usually interrupt-driven. That is, when an interrupt occurs, the executive stops what it is doing and processes the interrupt. After it processes the interrupt, the executive continues performing the application it was running before the interrupt occurred.

Real-time executives can also perform two very useful functions that do not (necessarily) rely upon outside interrupts: process scheduling and system services. Real-time systems can take care of scheduling your applications (tasks) so that you do not have to spend time setting up a priority handler; a real-time system can also take care of common system services, such as the timing functions.

5.1.2 DISTRIBUTED SYSTEMS

A distributed system is one in which the processing of information is distributed along a network of nodes connected by some type of communication link. These nodes are physical devices which form the system. Each node performs some kind of function specific to its place in the network.

5.2 MAJOR CHARACTERISTICS OF THE iRMX™ 51 EXECUTIVE

The iRMX 51 Executive is a real-time executive that you can use for system operations. This executive is designed to work both with a distributed network of nodes (such as the DCM system) and also with the 8051/8052 microcontrollers, as a stand-alone executive.

The major features of the iRMX 51 Executive include:

- Monitors and controls independent events which occur outside of the system.
- Communicates with a variety of devices.
- Provides scheduling for programs (tasks).
- Allows for inter-program (inter-task).
- Allows for inter-device communication when used as part of the DCM system.
- Provides a base upon which to run programs (tasks) written in several programming languages.
- Allows multiple tasks to share a processor.

The next section describes the iRMX 51 architecture; the architecture allows the iRMX 51 Executive to perform the functions listed in this section.

5.3 iRMX™ 51 ARCHITECTURE

This section discusses the way in which the iRMX 51 Executive architecture organizes system functions. The discussion of the iRMX 51 architecture consists of the following topics:

- Tasks.
- Sending messages.
- Using memory segments.

5.3.1 TASKS

A task is a program with specific characteristics that set it apart from conventional programs.

- Tasks can run asynchronously with respect to other tasks.
- Task can be synchronized with external events and with one another by means of interrupts and messages.

The following subsections expand upon these characteristics. The specific topics covered in the subsections are as follows:

- Multitasking.
- Task states.
- Task priorities and task preemption.
- Task size and task registers.

5.3.1.1 Multitasking

The iRMX 51 Executive uses multitasking to simplify the development of applications that process real-time events. The advantage to a real-time executive is the ability to process numerous events occurring at (seemingly) random times. These events are called asynchronous because they have no sequential relationship. Tasks can also execute events which are actually concurrent (meaning an event can occur while another is being processed). The ability of tasks to process events which are concurrent is based on a fact and on an assumption:

- The fact: tasks can share a processor.
- The assumption: none of the tasks will use the processor fully.

If you were to attempt to process asynchronous events without using an executive, you would have to use either the "single program" approach or the "separate modules" approach.

The single program approach attempts to handle asynchronous events in a sequential manner. This approach is often an awkward and inefficient way to deal with the asynchronous events. The separate modules approach attempts to handle asynchronous events by using separate programs to respond to various interrupts. This approach does not provide for inter-program communication and it allows only two levels of interrupts.

Multitasking is a technique which eliminates this confusion. Rather than writing a single program to process "X" events, you can write "X" programs, each of which processes a single event. This technique eliminates the need to monitor the order in which events occur.

Each of these "X" programs forms an iRMX 51 task. By multitasking, you simplify the process of building an application system. This allows you to build your system faster and at less expense. Furthermore, because of the one-to-one relationship between events and tasks, your system's code is less complex and easier to maintain.

5.3.1.2 Task States

An iRMX 51 task can be in any one of three states. The task states are asleep, ready, and running. These states are determined by how the tasks relate to events. An event is one of the following actions:

- An interrupt.
- A message (described in a later section).
- A time interval.

The following sections describe the states in which a task can be in relation to events.

5.3.1.2.1 THE ASLEEP STATE. A task is in the asleep state when it is waiting for an event to occur as the result of an RQ$WAIT system call. The event can be an interrupt, an interval cycle, or a message. In any case, the task remains asleep either until the event occurs, the "wait" times out, or until an RQ$DELETE$TASK system call deletes it. Refer to Chapter 6 for more information about iRMX 51 system calls.

5.3.1.2.2 THE READY STATE. A task is in the ready state when it is ready to run. The iRMX 51 Executive always selects the next running task from the ready group. A task can become ready for any of the following reasons:

- A task is created either at initialization or through an RQ$CREATE$TASK system call.
- A running task becomes a ready task if an event occurs which is associated with a higher priority task waiting for the event. The latter task, then, becomes the running task. The event can be an interrupt, a timeout, an interval cycle, a message, or a task creation.
- A task which is waiting for one or more events to occur (via RQ$WAIT) becomes ready when one of those events occurs. If the ready task has a higher priority than the running task, it becomes the running task.

Refer to Chapter 6 for more information about iRMX 51 system calls.

5.3.1.2.3 THE RUNNING STATE. A running task is the task that the processor is executing. The task remains in the running state until one of the following conditions occurs:

- The task uses RQ$WAIT to wait for one or more events to occur. If none of the events have occurred, and there is a non-zero timeout value, the running task becomes asleep. Then, the ready task with the highest priority becomes the running task.

- The task uses RQ$CREATE$TASK to create a task with a higher priority than itself. In this case, the newly created task becomes the running task, and the original running task becomes ready.

- The task uses RQ$SEND$MESSAGE to send a message to an on-chip task which is waiting for a message which has a higher priority than the running task.

- The task uses RQ$DELETE$TASK to delete itself. Then, the running task no longer exists and the highest priority ready task becomes the running task. If no ready task exists, the system becomes idle.

- A non-system clock interrupt occurs and the task associated with this interrupt is waiting for an interrupt event (as a result of the RQ$WAIT system call). Then, the latter task becomes the running task, and the previous running task becomes ready. Note that an interrupt event is always associated with a task that has a higher priority than the running task.

- A time-interval event occurs which is associated with a task that was waiting for this event. The task waiting for this event must have a higher priority than the running task. If this occurs, the latter task becomes the running task and the previous running task becomes ready.

- A timeout on an RQ$WAIT system call occurs; this timeout is associated with a task of a higher priority than the running task. Then, the task that was waiting becomes the running task and the previously running task becomes ready.

Refer to Chapter 6 for more information about iRMX 51 system calls. Figure 5-1 illustrates the relationships between the three task states.

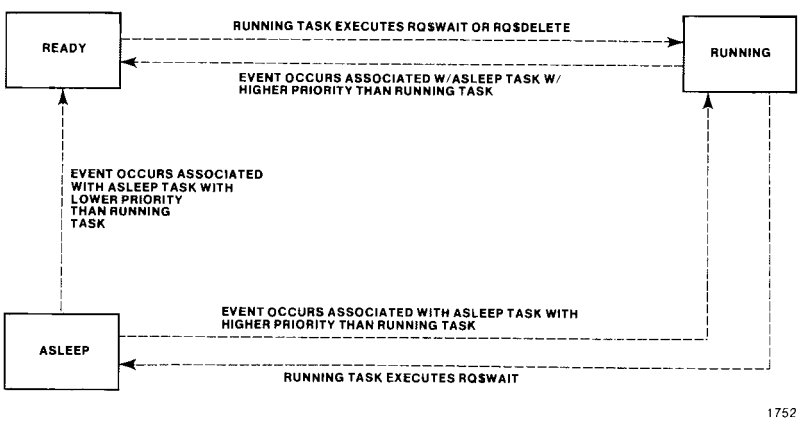


Figure 5-1. Relationships between Task States

5.3.1.3 Task Priority and Task Preemption

You assign the priority level of each task at configuration time when you specify the values in the Initial Task Descriptor (ITD). The iRMX 51 Executive uses the priority level to determine:

- Which interrupts to mask while a particular task is running. The iRMX 51 Executive masks interrupts associated with all tasks with the same or lower priority level than the running task.
- Which ready task to schedule as the next running task. The iRMX 51 Executive selects a task from the highest priority level which contains one or more ready tasks. Tasks are further selected from this group in a round-robin manner.

Preemption (with respect to tasks) is the act of a higher priority task causing the running task to return to the ready group; the higher priority task then becomes the running task. A running task can be preempted only when one of the following events occurs:

- An interrupt. Only a higher priority task waiting for an interrupt can preempt the running task (when the interrupt occurs).
- An iRMX 51 System Call. A task can be preempted when it makes a system call which causes a ready task's priority to change. The following system calls can raise the priority of a ready task: RQ$CREATE$TASK and RQ$SEND$MESSAGE. Refer to Chapter 6 for more information about iRMX 51 system calls.

Whenever the higher priority task (running) preempts a task, the preempted task will become the running task again when no higher priority task is in the ready or running states.

5.3.1.4 Task Size and Task Registers

The task size is limited only by the how much memory is available for code. Chapter 7 explains how to configure tasks and how to set up the Initial Task Descriptor (ITD).

The way in which tasks use registers is configurable, therefore, it is described in Chapter 7.

5.3.2 SENDING MESSAGES

The iRMX 51 Executive allows tasks to send messages between both tasks local to the processor and tasks residing on other processors. This section describes the following topics:

- Message types.
- iRMX 51 message structure.
- How to send messages to local and remote tasks.

5.3.2.1 Message Types

The iRMX 51 message types are based upon the BITBUS message format. These message types were designed to be compatible with the BITBUS interconnect. However, they are equally as easy to use on a stand-alone iRMX 51 system. Refer to the BITBUS section of the DISTRIBUTED CONTROL MODULES DATA BOOK for more information about the BITBUS message format.

You can use two types of iRMX 51 messages: order messages and reply messages. Tasks use order messages to send commands to other tasks; these commands are imbedded within the iRMX 51 message structure. Tasks use reply messages to respond to order messages received from other tasks; these responses are imbedded within the iRMX 51 message structure. Refer to a later section for more information about the iRMX 51 message structure.

5.3.2.2 Message Structure

Figure 5-2 defines the iRMX 51 message structure; the next section describes the parameters of the structure. This message structure is compatible with every message structure included in the DCM product.

```
Link                    WORD
Message_length          BYTE
Message_type            BIT
Src_ext                 BIT
Dest_ext                BIT
Trk                     BIT
Reserved                BIT(4)
Node_address            BYTE
Source_task_id          BIT(4)
Destination_task_id     BIT(4)
Command/Response        BYTE
Message_information     BYTE(message_length - 7);
```

Figure 5-2. iRMX™ 51 Message Structure

The parameters shown in Figure 5-2 are as follows:

Link — A two-byte parameter that the system uses for maintaining a list of messages.

Message_length (including header)	A byte value which specifies the total number of bytes in the message. The message is a seven-byte piece of information plus the number of bytes of user data. The maximum message size is configurable. Refer to Chapter 7 for more information on configuring the iRMX 51 Executive.
Message_type	A bit that indicates whether the message is an order or a reply. If the bit is not set (0), the message is an order; if the bit is set (1), the message is a reply. If the message is an order, the iRMX 51 Executive uses the destination task ID as the destination. If the message is a reply, it uses the source task ID as the destination.
Src_ext	A bit value that indicates whether the task which sends an order message resides on an extension (=1) or on a device (=0). Extensions are described in Chapter 12 of this manual.
Dest_ext	A bit value that indicates whether the task which receives an order message resides on an extension (=1) or on a device (=0). Extensions are described in Chapter 12 of this manual.
Trk	A bit value the system uses to track a message during a BITBUS transfer. You must set the trk to zero (0) before sending an order message.
Node_address	This parameter has different values depending upon where the message is delivered. For local messages, the value is 00H. For messages traveling over the parallel bus only, the parameter is "0FFH." For order message traveling from a master device (or its extension) to a slave device (or its extension), the parameter is the SDLC node address of the slave device. Refer to Chapter 12 for more information about extensions. Refer to the BITBUS section of the DISTRIBUTED CONTROL MODULES DATA BOOK for more information about the protocol.

Source_task_id	A 4-bit value order that tells the system which task sent the message. When the reply message is sent, the system uses this value for the destination of the reply message.
Destination_task_id	A 4-bit value order that tells the system the task to which the message should be sent. When the reply is sent, the system uses this value for the source in the reply message.
Command/Response	A byte field that the task can use for sending and receiving information. This parameter is predefined for the RAC function. Refer to Chapter 8 of this manual for more information about the RAC functions. The system uses the response part of this parameter to return system level errors during delivery. Appendix A lists the error messages and the corresponding hexadecimal values in the "literals" files.
Message_information	A seven-byte piece of information plus the number of bytes of user data. The maximum message size is configurable using RQSYSBUFSIZE in the iRMX 51 configuration file, RMX51A.CFG. If you are using the DCM controller firmware, the maximum message size is 20 bytes. Refer to Chapter 7 for more information on configuring the iRMX 51 Executive. Refer to Appendix A for the contents of the configuration file. This parameter has a fixed structure for messages destined for a RAC task. Refer to Chapter 8 for more information about the RAC functions.

5.3.2.3 Sending Message to Tasks on the Local Processor

You can exchange messages between tasks on the local processor by using the RQ$SEND$MESSAGE and the RQ$WAIT system calls described in Chapter 6. You must use these system calls and the message structure described in the previous section to get the information to the task.

5.3.2.4 Sending Messages to Tasks on Remote Devices

You can send messages to tasks on remote devices in the same manner that you send them to tasks on local devices, with one change. You must specify a non-zero node address in the message header. **Then, the iRMX 51 Executive automatically passes the message to Task 0 (the RAC task).**

Refer to Chapter 8 for more information about sending tasks to remote devices.

5.3.3 HOW TASKS USE MEMORY

Messages which must be sent to remote tasks have some restrictions on the way in which they use memory. There are no special instructions for using memory to send messages between local tasks.

All messages which are sent to tasks located on remote devices must be located in on-chip RAM. The iRMX 51 system calls, RQ$ALLOCATE and RQ$DEALLOCATE, supply and return memory in this space.

Order messages sent by the master device must be located in on-chip RAM. Incoming order messages received by slave devices are already in an on-chip system buffer. You can use this buffer to send the reply or you can allocate another system buffer. The number of on-chip buffers is limited by the message size, so each task must be sure to deallocate buffers they are no longer using.

5.3.4 SYSTEM VARIABLES

The system variables described in this section are important to you. You will see them in the iRMX 51 system calls described in Chapter 6. Normally, system variables are not important to the user; however, you need to know about the following two variables:

RQTASKID — The RQTASKID variable is a byte value which contains the task id of the running task. You can reference this field to obtain the source id when you send a message. NEVER WRITE TO THIS FIELD. The following examples explain how to use this variable:

```
MOV  A,RQTASKID     ;ASM51 usage

ACC = RQTASKID      /*PL/M 51 usage*/
```

RQCLOCKUNIT

The RQCLOCKUNIT system variable is a word value that defines a unit of time for the system clock. This value must be expressed as a negative number. In addition, this variable must be 1 millisecond for master nodes on the BITBUS interface. For example, if a processor is running at 12Mhz, a value of -1000 indicates a 1 millisecond unit of time. By altering the value of this variable, you can dynamically change the system clock rate.

If you wanted to set the clock time to 10 milliseconds, you could issue the following instructions:

```
MOV  RQCLOCKUNIT,#HIGH(-10,000)     ;ASM51
MOV  RQCLOCKUNIT+1,#LOW(-10,000)

RQCLOCKUNIT = -10,000               /*PL/M 51*/
```

***

CHAPTER 6
iRMX™ 51 SYSTEM CALLS

This chapter describes the iRMX 51 system calls in detail. It explains the services the calls provide along with how to use them. This chapter is organized so that you can read it once and get a basic understanding of the iRMX 51 system calls. You can then use the corner tabs for quick referencing of a particular system call.

6.1 USING iRMX™ 51 SYSTEM CALLS

The iRMX 51 system calls help you to perform a variety of system-related operations. This section describes the services the system calls provide, and it explains the conventions this chapter uses in both the PL/M 51 and the ASM51 examples.

6.1.1 SERVICES THE SYSTEM CALLS PROVIDE

The iRMX 51 system calls make it easy to perform a number of system services. For example, the iRMX 51 Executive provides system calls which take care of the following functions:

- Creating and deleting tasks.
- Passing messages between both tasks local to the processor and tasks residing on other processors.
- Allocating and deallocating system memory.
- Enabling and disabling interrupts within applications.
- Performing the timing function for the system.

The iRMX 51 system calls include other services; the system calls are explained in detail later in this chapter.

6.1.2 PL/M 51 CALLING SEQUENCE AND EXAMPLE

Each system call description contains a sequence which explains how to use the call in a PL/M 51 program. In addition, each call description includes an example of how to use the system call in a typical PL/M 51 application.

This manual includes dollar signs ($) in the system calls to increase readability. If you are using these calls in PL/M 51 applications, you can either include the dollar signs, or you can leave them out.

6.1.3 ASM51 CALLING SEQUENCE AND EXAMPLE

Each system call description contains a sequence which explains how to use the call in a ASM51 program. In addition, each call description includes an example of how to use the system call in a typical ASM51 application.

This manual includes dollar signs ($) in the system calls to increase readability. If you are using these calls in ASM51 applications, you must omit the dollar signs.

6.1.4 HOW SYSTEM CALLS USE REGISTERS

When you make a system call, the system destroys the contents of certain registers. The system destroys registers R0, R1, R6, and R7 upon completion of a system call. The system leaves registers R2, R3, R4, and R5 in tact unless another task is sharing the assigned register bank.

6.2 DETAILED SYSTEM CALL DESCRIPTIONS

This section presents detailed descriptions of the iRMX 51 system calls in alphabetical order. These descriptions use a notation to distinguish between the system call and the parameters. The system calls are shown in CAPITAL LETTERS and the parameters are shown in lower case characters. This section describes the parameters and how to use them in example programs.

Table 6-1 is a summary of the system call functions; it lists the system calls, their parameters, and a brief description of their functions.

Table 6-1. iRMX™ 51 System Call Summary

System Call Name	Parameters	Description
RQ$ALLOCATE	buffer$location	Allocates one block of on-chip RAM.
RQ$CREATE$TASK	task$descriptor$ptr task$id	Create and schedule a user task.
RQ$DEALLOCATE	buffer$location	Returns one block of on-chip RAM to the buffer pool.
RQ$DELETE$TASK	task$id status	Delete the specified task.
RQ$DISABLE$INTERRUPT	interrupt$vector	Temporarily disable an interrupt which was assigned to the calling task at initialization.
RQ$ENABLE$INTERRUPT	interrupt$vector	Enable an interrupt which was previously disabled by RQ$DISABLE$INTERRUPT
RQGETFUNCTION$IDS	functionidtbl$ptr	Gets the function IDs for the tasks which are currently in the system.
RQ$SEND$MESSAGE	message$ptr	Sends a message to the specified task.
RQSETINTERVAL	signal$time	Begins running a signal timer for the calling task.
RQ$WAIT	event$vector timeout message$ptr status	Waits for an interrupt, message, or time interval.

6.2.1 RQ$ALLOCATE

The RQ$ALLOCATE system call allocates one block of on-chip system RAM for tasks to use.

```
buffer$location = RQ$ALLOCATE;
```

6.2.1.1 Output Parameter

buffer$location	The byte address of the first byte of the block of on-chip memory that the system allocates for tasks to use. If no memory blocks are available, the system returns a "0" to "buffer$location." The system returns the value of "buffer$location" through the accumulator.

6.2.1.2 Description

The RQ$ALLOCATE system call instructs the system to allocate one block of on-chip system RAM for tasks to use.

6.2.1.3 PL/M 51 Calling Sequence

The PL/M 51 calling sequence for the RQ$ALLOCATE system call is as follows:

```
buffer$location = RQ$ALLOCATE;
```

6.2.1.4 ASM51 Calling Sequence

The ASM51 calling sequence for the RQ$ALLOCATE system call is as follows:

```
CALL    RQALLOCATE
MOV     buffer$location,A
```

6.2.1.4 Examples

Figure 6-1 contains an example of how to use the RQ$ALLOCATE system call in a PL/M 51 application.

```
/*****************************************************************/
/*
/* This procedure gets a buffer from the buffer pool.  It
/* continues to request a buffer until one becomes available.
/*
/*****************************************************************/

SAMPLE_ALLOCATE: PROCEDURE BYTE;

  DECLARE buffer_ptr BYTE;

    buffer_ptr = RQALLOCATE;
    DO WHILE (buffer_ptr = 0);
      buffer_ptr = RQALLOCATE;
      END;
    RETURN (buffer_ptr);

  END SAMPLE_ALLOCATE;
```

Figure 6-1. PL/M 51 Example for RQ$ALLOCATE

Figure 6-2 contains an example of how to use the RQ$ALLOCATE system call in an ASM51 application.

```
;*****************************************************************
;
; This procedure gets a buffer from the buffer pool.  It
; continues to request a buffer until one becomes available.
;
; This procedure has no calling parameters.
;
; The procedure returns with buffer_ptr in ACC
;
;*****************************************************************

SAMPLE_ALLOCATE:
          LCALL RQALLOCATE          ; Call RQALLOCATE to get buffer.
          JZ SAMPLE_ALLOCATE
                                    ; If ACC is not zero, then a
                                    ;  buffer was allocated and ACC
                                    ;  is its address.  Otherwise,
                                    ;  try another time.  Note,
                                    ;  this loop on allocate only
                                    ;  works if some other task
                                    ;  can preempt this task (i.e.
                                    ;  the other task is of higher
                                    ;  priority) and that task will
                                    ;  eventually deallocate a
                                    ;  buffer.

          RET                       ; Return with buffer ptr in ACC.
```

Figure 6-2. ASM51 Example for RQ$ALLOCATE

6.2.2. RQ$CREATE$TASK

The RQ$CREATE$TASK system call creates and schedules a new user task.

```
task$id = RQ$CREATE$TASK(task$descriptor$ptr);
```

6.2.2.1 Input Parameter

task$descriptor$ptr The address of the initial task descriptor for the task you are creating. Refer to Chapter 7 of this manual for more information about the initial task descriptor. You should initialize the DPTR register with this address before making the RQ$CREATE$TASK system call.

6.2.2.1 Output Parameter

task$id A byte value that the system uses to identify a task. The system returns the value to the A register. The possible values are as follows:

Value	Description
0-7	A valid task identification value.
EMAXTASKS	The system did not create the task because the maximum number of tasks already exists. Only eight tasks can exist in the system at any time.
E$REGS	The system did not create the task because the register bank requested is not available.
E$BUFFERS	The system did not create the task because no system buffers were available for the task stack space.

Refer to Appendix A for the hexadecimal values that correspond to the error messages.

6.2.2.3 Description

The RQ$CREATE$TASK system call creates a new user task with the attributes specified in the associated initial task descriptor. (Refer to Chapter 7 for more information about the initial task descriptor.) It then places the new task in the "ready" state and schedules it on a "round robin" basis with other ready tasks of the same priority level.

6.2.2.4 PL/M 51 Calling Sequence

The PL/M 51 calling sequence for the RQ$CREATE$TASK system call is as follows:

```
DPL = LOW(.initial$task$descriptor);
DPH = HIGH(.initial$task$descriptor);
task$id = RQ$CREATE$TASK;
```


6.2.2.5 ASM51 Calling Sequence

The ASM51 calling sequence for the RQ$CREATE$TASK system call is as follows:

```
MOV     DPTR,#initialtaskdescriptor
CALL    RQCREATETASK
MOV     taskid,A
```

6.2.2.6 Examples

Figure 6-3 contains an example of how to use the RQ$CREATE$TASK system call in a PL/M 51 application.

```
/*************************************************************/
/*
/* This procedure creates the task defined in ITD
/* and returns its assigned task identifier, if successful.
/* If not successful, a value of 0FFH is returned.
/*
/* ITD is a Initial Task Descriptor which is defined in
/* ASM 51 and assembled for linkage with this program.
/* PL/M 51 does not have the ability to provide relative
/* offset addressing.  The ITD must be coded in assembly
/* language, which does allow relative offset addressing.
/* See the assembly language example for the RQCREATETASK
/* call.
/*
/*************************************************************/

DECLARE ITD STRUCTURE
   (check_byte   WORD,
    start_offset WORD,
    stack_size   BYTE,
    function_id  BYTE,
    reg_priority BYTE,
    interrupts   WORD,
    next_link    WORD)
   EXTERNAL;

SAMPLE_CREATE: PROCEDURE BYTE;

  DECLARE task_id BYTE;

    DPL = LOW(.ITD);
    DPH = HIGH(.ITD);
    task_id = RQCREATETASK;
    IF (task_id > 7)
      THEN DO;                    /* error */
        RETURN (0FFH);
        END;
      ELSE DO;                    /* no error */
        RETURN (task_id);
        END;

  END SAMPLE_CREATE;
```

Figure 6-3. PL/M 51 Example for RQ$CREATE$TASK

Figure 6-4 contains an example of how to use the RQ$CREATE$TASK system call in an ASM51 application.

```
;*****************************************************************
;
; This procedure creates the task defined in ITD and returns its
; assigned task identifier, if successful.  If not successful,
; a value of 0FFH is returned.  This procedure has no calling
; parameters.  The procedure returns with status in ACC.
;
;*****************************************************************

CSEG

SAMPLE_CREATE:                          ; Procedure entry point

          MOV   DPTR,#ITD               ; Put the location of the ITD
          LCALL RQCREATETASK            ;  in DPTR and call RQCREATETASK

          CLR   C                       ; Determine if status returned
          SUBB  A,#08H                  ;  in ACC is between 0 and 7
          JC    CREATED                 ; If so, then return the value
                                        ;   in ACC
          MOV   A,#0FFH                 ; Otherwise return 0FFH in ACC
CREATED:
          RET                           ; Return with status in ACC


;
; This program uses the ITD Macro to link with this program.
;

CSEG

ITD:
       %ITD(newtask_pc,8,24,3,2,3,ITD+1)
                        ; newtask_pc = start of task code
                        ; 8 = stack size
                        ; 24 = function id
                        ; 3 = register bank specifications
                        ; 2 = task priority
                        ; 3 = interrupt vector
                        ; ITD+1 = next ITD (terminates initialization)

NEWTASK_PC: LJMP  NEWTASK               ; Jump to start of task

CSEG

NEWTASK:                                ; Example code for the new task

            LJMP  NEWTASK               ; End of new task code loop
```

Figure 6-4. ASM51 Example for RQ$CREATE TASK

The ITD macro used in this example is described in Chapter 7 of this manual.

6.2.3 RQ$DEALLOCATE

The RQ$DEALLOCATE system call releases one block of on-chip RAM for the system to use. This call frees a block of on-chip RAM that a task was using.

```
CALL RQ$DEALLOCATE(buffer$location);
```

6.2.3.1 Input Parameter

buffer$location	The byte address at the beginning of the buffer you wish to free for the system to use. The system uses register "A" to pass the value of "buffer$location."

6.2.3.2 Description

The RQ$DEALLOCATE system call deallocates one block of on-chip RAM set up for tasks to use, thus freeing this memory for the system to use. Typically, you use RQ$DEALLOCATE for deallocating RAM that was previously allocated with the RQ$ALLOCATE system call.

6.2.3.3 PL/M 51 Calling Sequence

The PL/M 51 calling sequence for the RQ$DEALLOCATE system call is as follows:

```
ACC = buffer$location;
CALL RQ$DEALLOCATE;
```

6.2.3.4 ASM51 Calling Sequence

The ASM51 calling sequence for the RQ$DEALLOCATE system call is as follows:

```
MOV    A,buffer$location
CALL   RQDEALLOCATE
```

6.2.3.5 Examples

Figure 6-5 contains an example of how to use the RQ$DEALLOCATE system call in a PL/M 51 application.

```
/*************************************************************/
/*
/* This procedure returns a buffer to the buffer pool.
/*
/*************************************************************/

SAMPLE_DEALLOCATE: PROCEDURE (buffer_ptr);

  DECLARE buffer_ptr BYTE;

  DECLARE buffer BASED buffer_ptr (20) BYTE IDATA;

    ACC = buffer_ptr;
    CALL RQDEALLOCATE;
    RETURN;

  END SAMPLE_DEALLOCATE;

END;
```

Figure 6-5. PL/M 51 Example for RQ$DEALLOCATE

Figure 6-6 contains an example of how to use the RQ$DEALLOCATE system call in an ASM51 application.

```
;*************************************************************
;
; This procedure returns a buffer to the buffer pool.
;
; This procedure is called with buffer_ptr in R0.
;
; The procedure has no result parameters.
;
;*************************************************************

SAMPLE_DEALLOCATE:

          MOV   A,R0                ; Put buffer address in ACC
          LCALL RQDEALLOCATE        ;  and call RQDEALLOCATE.
          RET                       ; Return, no parameters.

END
```

Figure 6-6. ASM51 Example for RQ$DEALLOCATE

6.2.4 RQ$DELETE$TASK

The RQ$DELETE$TASK system call deletes the specified task and disables all interrupts associated with that task.

```
status = RQ$DELETE$TASK(task$id);
```

6.2.4.1 Input Parameter

task$id	The identification value of the task you wish to delete. Valid values are 0 through 7 (inclusive) and they reside in the "A" register. Note that the "A" register should be initialized with the task$id to be deleted before calling RQ$DELETE$TASK.

6.2.4.2 Output Parameter

status — The byte value the system returns to the "A" register. The possible values and their descriptions are as follows:

Value	Description
E$OK(0)	The system deleted the task.
E$EXIST	The system did not delete the task; a task with the specified "task$id" does not exist.

Refer to Appendix A for the hexadecimal values that correspond to the error messages.

6.2.4.3 Description

The RQ$DELETE$TASK system call deletes the task you specify. It also disables all interrupts associated with that task. The system does not recognize a deleted task's task ID until a newly created task reuses it.

6.2.4.4 PL/M 51 Calling Sequence

The PL/M 51 calling sequence for the RQ$DELETE$TASK system call is as follows:

```
ACC = task$id;
status = RQ$DELETE$TASK:
```

6.2.4.5 ASM51 Calling Sequence

The ASM51 calling sequence for the RQ$DELETE$TASK system call is as follows:

```
MOV    A,task$id
CALL   RQDELETETASK
MOV    status,A
```


6.2.4.6 Examples

Figure 6-7 contains an example of how to use the RQ$DELETE$TASK system call in a PL/M 51 application.

```
/*****************************************************************/
/*
/* This procedure deletes the task whose task identifier
/* is task_id and returns 0 if successful.
/* If not successful, a value of 0FFH is returned.
/*
/*****************************************************************/

SAMPLE_DELETE: PROCEDURE (task_id) BYTE;

  DECLARE task_id BYTE;

  DECLARE status BYTE;

    ACC = task_id;
    status = RQDELETETASK;
    IF (status <> 0)
      THEN DO;                    /* error */
        RETURN (0FFH);
        END;
      ELSE DO;                    /* no error */
        RETURN (0);
        END;

  END SAMPLE_DELETE;
```

Figure 6-7. PL/M 51 Example for RQ$DELETE$TASK

Figure 6-8 contains an example of how to use the RQ$DELETE$TASK system call in an ASM51 application.

```
;*****************************************************************
;
; This procedure deletes the task whose task identifier
; is task_id and returns 0 if successful.
; If not successful, a value of 0FFH is returned.
;
; This procedure is called with task_id in R0
;
; The procedure returns with status in ACC
;
;*****************************************************************

SAMPLE_DELETE:
          MOV   ACC,R0                ; Put task_id in ACC and
          LCALL RQDELETETASK          ;  call RQDELETETASK
          JZ    DELETED               ; If not successful,
          MOV   A,#0FFH               ;   return 0FFH
DELETED:
          RET
```

Figure 6-8. ASM51 Example for RQ$DELETE$TASK

6.2.5 RQ$DISABLE$INTERRUPT

The RQ$DISABLE$INTERRUPT task temporarily disables an interrupt at all priority levels.

```
CALL RQ$DISABLE$INTERRUPT(interrupt$number);
```

6.2.5.1 Input Parameter

interrupt$number — A byte value between 0H and 0FH which specifies the interrupt source you wish to disable. The interrupt numbers and the corresponding interrupt sources are as follows:

Interrupt Number	Interrupt Source
00H	external request 0
01H	timer 0 - system clock
02H	external request 1
03H	timer 1 - baud rate
04H	internal serial port 1
05H	reserved
06H	reserved
07H	reserved
08H	reserved
09H	reserved
0AH	reserved
0BH	reserved
0CH	reserved
0DH	reserved
0EH	reserved
0FH	reserved

The caller passes the "interrupt$number" parameter through the "A" register.

CAUTION

You should not disable timer 0 (under normal circumstances) because it runs the system clock.

6.2.5.2 Description

The RQ$DISABLE$INTERRUPT task disables an interrupt at all priority levels until RQ$ENABLE$INTERRUPT re-enables the interrupt. The system disables the specified interrupt immediately upon return from this call.

You can use the system call RQ$ENABLE$INTERRUPT to reactivate the interrupt.

6.2.5.3 PL/M 51 Calling Sequence

The PL/M 51 calling sequence for the RQ$DISABLE$INTERRUPT system call is as follows:

```
ACC = interrupt$number;
CALL RQ$DISABLE$INTERRUPT;
```

6.2.5.4 ASM51 Calling Sequence

The ASM51 calling sequence for the RQ$DISABLE$INTERRUPT system call is as follows:

```
MOV    A,interrupt$number
CALL   RQDISABLEINTERRUPT
```

6.2.5.5 Examples

Figure 6-9 contains an example of how to use the RQ$DISABLE$INTERRUPT system call in a PL/M 51 application.

```
/*****************************************************************/
/*
/* This procedure disables the timer 1 interrupt level.
/*
/*****************************************************************/

SAMPLE_DISABLE: PROCEDURE;

    ACC = 03H;                  /* Timer 1 interrupt */
    CALL RQDISABLEINTERRUPT;
    RETURN;

  END SAMPLE_DISABLE;
```

Figure 6-9. PL/M 51 Example for RQ$DISABLE$INTERRUPT

Figure 6-10 contains an example of how to use the RQ$DISABLE$INTERRUPT system call in an ASM51 application.

```
;*****************************************************************
;
; This procedure disables the timer 1 interrupt level.
;
; This procedure has no calling parameters.
;
; The procedure has no result parameters.
;
;*****************************************************************

SAMPLE_DISABLE:
          MOV   A,#03H                ; Put 03 in ACC for timer 1
          LCALL RQDISABLEINTERRUPT    ;  interrupt and call
                                      ;  RQDISABLEINTERRUPT
          RET                         ; Return, no parameters.
```

Figure 6-10. ASM51 Example for RQ$DISABLE$INTERRUPT

6.2.6 RQ$ENABLE$INTERRUPT

The RQ$ENABLE$INTERRUPT system call re-enables an interrupt which was previously disabled by the RQ$DISABLE$INTERRUPT system call.

```
CALL RQ$ENABLE$INTERRUPT(interrupt$number);
```

6.2.6.1 Input Parameter

interrupt$number A byte value between 0H and 0FH which specifies the interrupt source you wish to reactivate. The interrupt numbers and the corresponding interrupt sources are as follows:

Interrupt Number	Interrupt Source
00H	external request 0
01H	timer 0
02H	external request 1
03H	timer 1
04H	internal serial port 1
05H	reserved
06H	reserved
07H	reserved
08H	reserved
09H	reserved
0AH	reserved
0BH	reserved
0CH	reserved
0DH	reserved
0EH	reserved
0FH	reserved

The user passes the "interrupt$number" parameter through the "A" register.

6.2.6.2 Description

The RQ$ENABLE$INTERRUPT system call allows a task to re-enables an interrupt which RQ$DISABLE$INTERRUPT previously disabled.. This system call cannot enable an interrupt that was not originally disabled by the RQ$DISABLE$INTERRUPT system call.

6.2.6.3 PL/M 51 Calling Sequence

The PL/M 51 calling sequence for the RQ$ENABLE$INTERRUPT system call is as follows:

```
ACC = interrupt$number;
CALL RQ$ENABLE$INTERRUPT;
```

6.2.6.4 ASM51 Calling Sequence

The ASM51 calling sequence for the RQ$ENABLE$INTERRUPT system call is as follows:

```
MOV    A,interrupt$number
CALL   RQENABLEINTERRUPT
```


6.2.6.5 Examples

Figure 6-11 contains an example of how to use the RQ$ENABLE$INTERRUPT system call in a PL/M 51 application.

```
/*****************************************************************/
/*
/* This procedure re-enables the timer 1 interrupt level.
/*
/*****************************************************************/

SAMPLE_ENABLE: PROCEDURE;

    ACC = 03H;                  /* Timer 1 interrupt */
    CALL RQENABLEINTERRUPT;
    RETURN;

  END SAMPLE_ENABLE;
```

Figure 6-11. PL/M 51 Example for RQ$ENABLE$INTERRUPT

Figure 6-12 contains an example of how to use the RQ$ENABLE$INTERRUPT system call in an ASM51 application.

```
;*******************************************************
;
; This procedure enables the timer 1 interrupt level.
;
; This procedure has no calling parameters.
;
; The procedure has no result parameters.
;
;*******************************************************

SAMPLE_ENABLE:
          MOV    A,#03H                ; Put 03 in ACC for timer 1
          LCALL RQENABLEINTERRUPT      ;  interrupt and call
                                       ;  RQENABLEINTERRUPT
          RET                          ; Return, no parameters.
```

Figure 6-12. ASM51 Example for RQ$ENABLE$INTERRUPT

6.2.7 RQGETFUNCTION$IDS

The RQGETFUNCTION$IDS system call retrieves the functional identification values that you can arbitrarily assign to the tasks which are currently in the system. (Tasks do not require function ids; they simply allow you to keep track of them by function.)

You can assign functional IDs when you define the Initial Task Descriptor (IDT); refer to Chapter 7 for more information about defining the fields of the IDT. Refer to the BITBUS section of the DISTRIBUTED CONTROL MODULES DATA BOOK for more information about the function IDs on the DCM system.

```
CALL RQ$GET$FUNCTION$IDS(function$id$tbl$ptr);
```


6.2.7.1 Input Parameter

functionidtbl$ptr A pointer to an 8-byte block of memory where you want the system to place the functional identification values for the tasks currently in the system.

If you choose a pointer value less than 256, the system installs the functional IDs in on-chip memory. If you choose a pointer value of 256 or greater, the system installs the functional IDs in off-chip memory.

Each entry in the block of memory corresponds positionally to a functional ID value. An entry of 0H means that no task with the functional ID exists currently. An entry of 0FFH means that a task with the specified functional ID exists, but it is significant only in local (on-chip) functions. For example, in the following table, Task 0 has a functional ID value of seven, Task 1 has a functional ID of two, Task 2 has a functional ID of zero and thus does not exist, and Task 3 is a local on-chip function. An example function id table is as follows:

Task IDs	Function IDs
Task 0	7
Task 1	2
Task 2	0
Task 3	0FFH

The user task passes this pointer through the Data Pointer (DPTR) register. You are responsible for assigning function IDs; refer to Chapter 7 for more information about ITDs and function IDs.

6.2.7.2 Description

The RQGETFUNCTION$IDS system call retrieves the functional identification values (functional IDs) that you can assign to the tasks which are currently in the system.

You can assign functional IDs when you define the Initial Task Descriptor (IDT). By assigning a function ID to a task, you inform the system of that tasks function and thus allow the system to take advantage of the task's service. Refer to Chapter 7 for more information about defining the fields of the IDT.

6.2.7.3 PL/M 51 Calling Sequence

The PL/M 51 calling sequence for the RQGETFUNCTION$IDS system call is as follows:

```
DPL = LOW(.function$id$table);
DPH = HIGH(.function$id$table);
CALL RQ$GET$FUNCTION$IDS;
```

6.2.7.4 ASM51 Calling Sequence

The ASM51 calling sequence for the RQGETFUNCTION$IDS system call is as follows:

```
MOV     DPTR,#function$id$table
CALL    RQGETFUCNTIONIDS
```

6.2.7.5 Examples

Figure 6-13 contains an example of how to use the RQGETFUNCTION$IDS system call in a PL/M 51 application.

```
/*****************************************************************/
/*
/* This procedure finds the task identifier for a given
/* function identifier.  It gets the function identifiers
/* for the up to eight tasks which are currently active.
/* Then the list is searched for the specific function
/* identifier wanted.  If found, the respective task
/* identifier is returned. Otherwise, 0FFH is returned.
/*
/*****************************************************************/

SAMPLE_GET: PROCEDURE (function_id) BYTE;

  DECLARE function_id BYTE;
  DECLARE function_id_table (8) BYTE;
  DECLARE task_id BYTE;

    DPL = LOW(.function_id_table);
    DPH = HIGH(.function_id_table);
    CALL RQGETFUNCTIONIDS;
    DO task_id = 0 TO 7;
      IF (function_id = function_id_table(task_id))
        THEN RETURN (task_id);
      END;
    RETURN (0FFH);

  END SAMPLE_GET;
```

Figure 6-13. PL/M 51 Example for RQGETFUNCTION$IDS

Figure 6-14 contains an example of how to use the RQGETFUNCTION$IDS system call in an ASM51 application.

```
;*****************************************************************
;
; This procedure finds the task identifier for a given
; function identifier.  It gets the function identifiers
; for the up to eight tasks which are currently active.
; Then the list is searched for the specific function
; identifier wanted.  If found, the respective task
; identifier is returned. Otherwise, 0FFH is returned.
;
; This procedure is called with function_id in R0.
;
; The procedure returns with task_id or 0FFH in ACC.
;
;*****************************************************************

XSEG

FUNCTION_ID_TABLE: DS 8              ; Function_ID_table is in
                                     ; off-chip memory.

CSEG

SAMPLE_GET:
          MOV   A, R0                ; Save the function_id
          MOV   R2, A                ;  in R2 for call
                                     ; Put the address of the function
                                     ;  id table in DPTR and call
                                     ;  RQGETFUNCTIONIDS
          MOV   DPTR,#FUNCTION_ID_TABLE
          LCALL RQGETFUNCTIONIDS
                                     ; Put the address of the function
                                     ;  id table back in DPTR
          MOV   DPTR,#FUNCTION_ID_TABLE
          MOV   A, DPL               ;  and offset to the last entry
          ADD   A, #7
          MOV   DPL, A
          MOV   R0,#08H              ; Put task_id index value in R0
                                     ;  for indexing function id table
SEARCH:
          MOV   A, R0                ; See if all entries have been
          JZ    NOT_FOUND            ;  checked.  If so, then jump out
                                     ;  of the loop.
          MOVX  A,@DPTR              ; If not, get current entry
                                     ;  to function id
          DEC   DPL                  ; Set DPTR to next entry
          CJNE  A,R2,SEARCH          ; Compare current entry function
          JMP   FOUND                ;  id to function id.  If they
                                     ;  are different then repeat
```

Figure 6-14. ASM51 Example for RQGETFUNCTION$IDS

```
                                    ;  for next entry.  If
                                    ;  they are the same, then return
            MOV     A,R0            ;  the index as task_id in ACC.
            RET

NOT_FOUND:
            MOV     A,#0FFH         ; If no function id match is
            RET                     ; found then return 0FFH in ACC.
```

Figure 6-14. ASM51 Example for RQGETFUNCTION$IDS (continued)

6.2.8 RQ$SEND$MESSAGE

The RQ$SEND$MESSAGE system call sends a message to the specified task.

```
CALL RQ$SEND$MESSAGE(message$ptr);
```

6.2.8.1 Input Parameter

message$ptr	A pointer to the message you wish to send to the task you specify in the message header. (Refer to Chapter 5 for more information about the structure of messages.) If you choose a pointer value less than 256, the system regards the pointer as pointing to on-chip memory. If you choose a pointer value of 256 or greater, the system regards the pointer as pointing to off-chip memory. You pass this pointer through the Data Pointer (DPTR) register.

6.2.8.2 Description

The RQ$SEND$MESSAGE system call sends a message to the specified task. You specify the task through the message header. Refer to Chapter 5 for more information about the iRMX 51 message structure.

If the message is an order, the system gets the receiving task's address from the receiving task's fields. If the message is a reply, the system gets the receiving task's address from the sending task's fields.

6.2.8.3 PL/M 51 Calling Sequence

The PL/M 51 calling sequence for the RQ$SEND$MESSAGE system call is as follows:

```
DPL = LOW(.message);
DPH = HIGH(.message);
CALL RQ$SEND$MESSAGE;
```

6.2.8.4 ASM51 Calling Sequence

The ASM51 calling sequence for the RQ$SEND$MESSAGE system call is as follows:

```
MOV     DPTR,#message
CALL    RQSENDMESSAGE
```

6.2.8.5 Examples

Figure 6-15 contains an example of how to use the RQ$SEND$MESSAGE system call in a PL/M 51 application.

```
/*************************************************************/
/*
/* This procedure sends a message to a destination task
/* on the same component with a command value of 12 and
/* a single data value of 4.  It determines the source task
/* from RQTASKID, which contains the task id for the current
/* running task, i.e. the calling task.
/*
/*************************************************************/

SAMPLE_SEND: PROCEDURE (dest_task_id);

  DECLARE dest_task_id BYTE;

  DECLARE message_ptr WORD;
  DECLARE message BASED message_ptr STRUCTURE
    (link WORD,
     length BYTE,
     route BYTE,
     node BYTE,
     tasks BYTE,
     cmd_rsp BYTE,
     params (1) BYTE) IDATA;

    message.length = 8;
    message.route = 0;
    message.node = 0;
    message.tasks = SHL(RQTASKID,4) OR dest_task_id;
    message.cmd_rsp = 12;
    message.params (0) = 4;

    DPL = LOW(message_ptr);
    DPH = HIGH(message_ptr);
    CALL RQSENDMESSAGE;
    RETURN;

  END SAMPLE_SEND;
```

Figure 6-15. PL/M 51 Example for RQ$SEND$MESSAGE

Figure 6-16 contains an example of how to use the RQ$SEND$MESSAGE system call in an ASM51 application.

```
;*****************************************************************
;
; This procedure sends a message to a destination task
; on the same component with a command value of 12 and
; a single data value of 4.
;
; This procedure is called with dest_task_id in R1.
;
; The procedure has no result parameters.
;
;*****************************************************************

DSEG

MESSAGE: DS 20                          ; on-chip message buffer

CSEG

SAMPLE_SEND:
          MOV   R0,#MESSAGE             ; Set up R0 to use as index to
          INC   R0                      ;  message fields and offset to
          INC   R0                      ;  the length field.
          MOV   @R0,#08H                ; Put a length value of 8 in
                                        ;  field
          INC   R0                      ; Offset index to route field.
          MOV   @R0,#00H                ; Put 0, i.e. Order type, Source
                                        ;  not extended, Destination not
                                        ;  extended, Trk Bit zeroed, in
                                        ;  the route field.
          INC   R0                      ; Offset index to node field
          MOV   @R0,#00H                ; Put 0, i.e. local node, in
                                        ;  node field.
          INC   R0                      ; Offset index to tasks field
          MOV   A, RQTASKID             ; Get the current running task
                                        ;  id (task id for this task)
          SWAP  A                       ; Set the source task id to
          ANL   A,#0F0H                 ;  this_task_id and the
                                        ;  destination
          ORL   A,R1                    ;  task id to dest_task_id in
          MOV   @R0,A                   ;  the tasks field.
          INC   R0                      ; Offset index to the cmd_rsp
                                        ;  field
          MOV   @R0,#0CH                ; Set the cmd_rsp field value to
                                        :  12
          INC   R0                      ; Offset to the first data field
          MOV   @R0,#04H                ; Set the first data field to 4
          MOV   DPTR, #MESSAGE          ; Put the address of the message
          LCALL RQSENDMESSAGE           ;  in DPTR and call RQSENDMESSAGE
          RET                           ; Return
```

Figure 6-16. ASM51 Example for RQ$SEND$MESSAGE

6.2.9 RQSETINTERVAL

The RQSETINTERVAL system call sets up and runs a signal timer for the task.

```
CALL RQ$SET$INTERVAL(signal$time);
```

6.2.9.1 Input Parameter

signal$time	A byte value for the amount of time you want the system to wait between signals for this task. A "signal$time" of zero (0) stops the signal. The user task passes the byte value through the accumulator.

6.2.9.2 Description

The RQSETINTERVAL system call sets up and runs a signal timer for the task. If a signal timer already exists for the task, RQSETINTERVAL resets the signal at the time of the call. (The signals occur ever "signal$time" units.)

6.2.9.3 PL/M 51 Calling Sequence

The PL/M 51 calling sequence for the RQSETINTERVAL system call is as follows:

```
ACC = signal$time;
CALL RQ$SET$INTERVAL;
```

6.2.9.4 ASM51 Calling Sequence

The ASM51 calling sequence for the RQSETINTERVAL system call is as follows:

```
MOV    A,signal$time
CALL   RQSETINTERVAL
```

6.2.9.5 Examples

Figure 6-17 contains an example of how to use the RQSETINTERVAL system call in a PL/M 51 application.

```
/*********************************************************/
/*
/* This procedure sets the interval time for this task to
/* a specified number of time units.
/*
/*********************************************************/

SAMPLE_SET: PROCEDURE (interval_time);

  DECLARE interval_time BYTE;

    ACC = interval_time;
    CALL RQSETINTERVAL;
    RETURN;

  END SAMPLE_SET;
```

Figure 6-17. PL/M 51 Example for RQSETINTERVAL

Figure 6-18 contains an example of how to use the RQSETINTERVAL system call in an ASM51 application.

```
;*****************************************************************
;
; This procedure sets the interval time for this task to
; a specified number of time units.
;
; This procedure is called with interval_time in R0.
;
; The procedure has no result parameters.
;
;*****************************************************************

SAMPLE_SET:
          MOV   A,R0                ;Put the interval_time in ACC
          LCALL RQSETINTERVAL       ; and call RQSETINTERVAL
          RET                       ; Return
```

Figure 6-18. ASM51 Example for RQSETINTERVAL

6.2.10 RQ$WAIT

The RQ$WAIT system call allows a task to wait for a signal, an interrupt, a message, or a timeout.

```
status,message$ptr = RQ$WAIT(event$vector,timeout);
```

6.2.10.1 Input Parameters

event$vector — A byte value that indicates the events for which the task is waiting. The "event$vector" is in the following form:

XXXXXabc

where "X" is a system value that you should ignore and "abc" are values that specify the type of events for which the task will wait.

If "a" is set to one (1), the task waits for a timer interval. If "b" is set to one (1), the task waits for an interrupt. If "c" is set to one (1), the task waits for a message. The task does not wait for an event if its corresponding bit is set to zero (0).

You place the "event$vector" into the "A" register before making the call.

timeout — A byte value (0 to 0FFH) that specifies the amount of number of time units that the task will wait before it returns the "timeout" status.

If you choose a byte value of zero (0), the task does not wait at all. If you choose a byte value of 0FFH, the task waits for an interrupt, a signal, or a message forever.

You place this value into the "B" register before making the call.

6.2.10.2 Output Parameters

status — A byte value that indicates which event (or events) caused the task to resume its operation after an RQ$WAIT system call. The status can be one of the following values, or it can be any "OR'd" combination of two of more of the following values:

Event	Valued "OR'd" into Status
timeout	0000 0000
time interval	0000 0100
interrupt	XXXX 0010*
message	0000 0001

*For interrupts, "XXXX" indicates which of 16 interrupt sources caused the interrupt. The system returns the status in the "A" register.

NOTE

The interrupt number the system returns to RQ$WAIT reflects the most recent interrupt to occur. The only case in which more than one interrupt occurs (for a task) is between RQ$WAIT calls that do not specify that the task must wait for interrupt events.

message$ptr — A pointer to the message if "status" bit 2 is set. The system places this value in registers "R6" and "R7." If the value of "message$pointer" is less than 256, the message is in on-chip RAM. If the value of "message$pointer" is greater than or equal to 256, the message is in off-chip RAM.

6.2.10.3 Description

The RQ$WAIT system call allows a task to wait for a signal, an interrupt, or a message. If one of these events has already taken place when the task calls RQ$WAIT, the system immediately returns control to the task and sends the proper status.

The task passes the "timeout" indicated in the system call to the system. You must set the signal and enable the appropriate interrupts before you use the RQ$WAIT call.

6.2.10.4 PL/M 51 Calling Sequence

The PL/M 51 calling sequence for the RQ$WAIT system call is as follows:

```
ACC = event$vector;
B = timeout;
message$ptr = RQ$WAIT;
status = ACC;
```

6.2.10.5 ASM51 Calling Sequence

The ASM51 calling sequence for the RQ$WAIT system call is as follows:

```
MOV     A,eventvector
MOV     B,timeout
CALL    RQWAIT
MOV     status,A
MOV     messageptr+1,R7
MOV     messageptr,R6
```

6.2.10.6 Examples

Figure 6-19 contains an example of how to use the RQ$WAIT system call in a PL/M 51 application.


```
/*****************************************************************/
/*
/* This procedure waits for 20 time units for a message to
/* arrive.  If a timeout occurs, a 0FFH status is returned.
/* Otherwise, a 0 status is returned with the message
/* location.
/*
/*****************************************************************/

SAMPLE_WAIT: PROCEDURE (message_ptr) BYTE;

  DECLARE message_ptr WORD;

  DECLARE message BASED message_ptr STRUCTURE
    (link WORD,
     length BYTE,
     route BYTE,
     node BYTE,
     tasks BYTE,
     cmd_rsp BYTE,
     params (13) BYTE) IDATA;

  DECLARE status BYTE;

    ACC = 1;                    /* wait for message or timeout */
    B = 20;                     /* timeout set to 20 units */
    message_ptr = RQWAIT;
    status = ACC;
    IF (status = 0)
      THEN DO;                  /* timeout */
        RETURN (0FFH);
        END;
```

Figure 6-19. PL/M 51 Example for RQ$WAIT

```
        ELSE DO;                    /* message */
          RETURN (0);
          END;

    END SAMPLE_WAIT;
```

Figure 6-19. PL/M 51 Example for RQ$WAIT (continued)

Figure 6-20 contains an example of how to use the RQ$WAIT system call in an ASM51 application.

```
;*****************************************************************
;
; This procedure waits for 20 time units for a message to
; arrive.  If a timeout occurs, a 0FFH status is returned.
; Otherwise, a 0 status is returned with the message
; location.
;
; This procedure has no calling parameters.
;
; The procedure returns with status in ACC and
; message_ptr in R6 and R7.
;
;*****************************************************************

SAMPLE_WAIT:
          MOV   A,#01H              ; Set the event type for wait
                                    ;  to 1, i.e. message, in ACC.
          MOV   B,#14H              ; Set the timeout to 20 units.
          LCALL RQWAIT              ;  and call RQWAIT
          CJNE  A,#00H,RECEIVED     ; Upon return, check the status.
                                    ;  if zero, then a timeout has
          MOV   A,#0FFH             ;  occurred.  Set the return
          RET                       ;  status to 0FFH in ACC and
                                    :  return.
RECEIVED:                           ; Otherwise, the status must be
                                    :  1.
          MOV   A,#00H              ; The message pointer is in DPTR.
          RET                       ; Return 0 status in ACC.
```

Figure 6-20. ASM51 Example for RQ$WAIT

***

CHAPTER 7 — CONFIGURATION OF THE STAND-ALONE iRMX™ 51 SYSTEM AND THE DCM CONTROLLER FIRMWARE

This chapter describes how to configure both a stand-alone iRMX 51 Executive and the DCM controller firmware. You can use this chapter to help you tailor the iRMX 51 Executive to your application. This chapter explains how to link and locate your system; it describes this process both for using the iRMX 51 Executive as a stand-alone system and for using the iRMX 51 Executive (in preconfigured firmware) as part of the DCM system.

7.1 INITIAL TASK DESCRIPTOR

The Initial Task Descriptor (ITD) structure allows you to specify the original attributes of a task. You define this structure at configuration time for the original task list and when you make run-time RQ$CREATE$TASK system calls.

Figure 7-1 defines the ITD structure.

```
Pattern                 WORD    DATA('1010101001010101B'),
Initial_PC              WORD,
Stack_length            BYTE,
Function_id             BYTE,
Register_bank           BIT(4),
Priority                BIT(4),
Interrupt_vector        WORD,
Next_ITD                WORD;
```

Figure 7-1. ITD Structure

The ITD parameters in Figure 7-1 are as follows:

Pattern	A 16-bit value which indicates that this structure is an Initial Task Descriptor. The system stops initializing the task when the "next_ITD" field points to an invalid pattern.
Initial_PC	The 16-bit address (relative to the beginning of the ITD) of the first instruction the task will execute.

For example, if the task's code begins 30 bytes before the ITD, the value is 30; if the code starts 30 bytes after the ITD, the value is -30 or 0FFE2H.

Stack_length

A value that specifies the number of bytes of system RAM that you want to allocate for the task's stack. You should specify at least four bytes.

If you are dynamically creating the task by using RQ$CREATE$TASK, the system ignores this parameter and allocates one system buffer for the task's stack. Therefore, if the the system buffer size is 20 bytes, a dynamically created task will have a 20 byte stack.

Function_id

A byte value between 1 and 255 which associates a task with a function. You should assign the value of 255 (0FFH) to tasks:

- That should not be associated with a function outside of the on-chip environment.
- If you do not want to publicize the function of the task.

The value of one (1) is reserved for the RAC task (Task 0). Refer to Chapter 5 for more information about the RAC task.

If you should assign duplicate function_ids, the system will terminate task initialization. This action prevents a task from being initialized twice; it also prevents the following situations from occurring:

- The next-ITD pointer pointing to memory that does not exist.
- Memory wrap-around.

It is permissible to have duplicate FFH function_ids, but it is your responsibility to prevent one of these tasks from being initialized twice.

Register_bank

A half-byte value which assigns one of the four register banks to this task. The possible bit values are as follows:

00xx Any register bank will work, where "xx" = the system does not care.

01rr Use register bank "rr," where "rr" = 0 to 3. You should specify this option when you are coding in PL/M 51.

The system allocates the register bank according to the following algorithm:

- If you do not request a specific register bank, the system selects the lowest number (unused) register bank. If no banks are free, an error condition occurs.
- If you request a specific register bank which is already associated with the specified priority level (or it is presently unused by a task) the system selects it. Otherwise, an error condition occurs.

There are two cases when the system cannot meet a request for a register bank. They are as follows:

- You do not request a specific register bank and all four register banks are associated with priority levels other than the requesting task's priority level.
- You request a specific register bank and that bank is already associated with a priority level different than that of the requesting task's.

You can reduce the risk of unsatisfied register requests by ordering ITD's to initialize tasks that request specific register banks first.

Priority — A half-byte value between 1 and 4 that specifies the task's priority level, where 4 is the highest priority level.

Interrupt_vector — A two-byte value that specifies which interrupt you want to be associated with the task. This value's format is as follows:

XXXXXXXX XXXabcXd

where "X" is a system value that you should ignore and bits "a," "b," "c," and "d" are shown in Table 7-1.

Table 7-1. Interrupt Vector Values

Interrupt Source	Assigned to a Task?	
	Yes	No
Internal Serial Port 1	a = 1	a = 0
Timer 1	b = 1	b = 0
External Request 1	c = 1	c = 0
External Request 0	d = 1	d = 0

Next_ITD — A word value that contains the address (relative to the beginning of the ITD) of the next ITD in the linked-list. You can terminate the ITD list by specifying any relative address which points to an invalid bit pattern. Pointing to ITD + 1 always terminates initialization.

7.2 CONFIGURING THE iRMX™ 51 EXECUTIVE AS A STAND-ALONE SYSTEM

This section describes how to configure the iRMX 51 Executive as a stand-alone system. These instructions apply to the iRMX 51 Executive as it is supplied on the iRMX 51 Executive diskette. (A later section describes how to configure the DCM controller firmware.) Configuring the iRMX 51 Executive involves the following steps:

1. Specifying the initial set of task descriptors.
2. Specifying the configuration values.
3. Linking the system with the user-supplied modules.

This section describes how to configure your iRMX 51 Executive by using an example configuration which explains the configuration steps. The configuration example consists of an iRMX 51 system with three user tasks which are contained in three source modules: two ASM51 modules and one PL/M 51 module.

7.2.1 iRMX™ 51 CONFIGURATION EXAMPLE (MODULE 1)

Figure 7-2 lists the contents of "module 1" of the iRMX 51 configuration example. It contains the ITDs for the three tasks in the system.

```
$INCLUDE(:fx:rmx51a.ext)
$INCLUDE(:fx:rmx51a.lit)
$INCLUDE(:fx:rmx51a.mac)
$INCLUDE(:fx:rmx51a.cfg)

    CSEG

EXTRN CODE(Task1, Task2)

ITD1:
%ITD(Task1_Entry, 8, 002, 000B, 4, 10101B, ITD2)
ITD2:
%ITD(Task2_Entry, 4, 200, 111B, 3, 00000B, ITD3)
ITD3:
%ITD(Task3_Entry, 4, 145, 000B, 3, 01000B, ITD3+1)

Task1_Entry:
    AJMP Task1

Task2_Entry:
    AJMP Task2

Task3_Entry:
    MOV     A,#010B             ; wait indefinitely for Timer 1
    MOV     B,#0FFH             ; interrupt
    CALL    RQWAIT

    •
    •                           performs interrupt processing
    •

JMP Task3_Entry

END
```

Figure 7-2. iRMX™ 51 Executive Configuration Example (Module 1)

This module uses the ASM51 macro "%ITD" to simplify how you specify the task descriptor fields. The %ITD macro is defined in the file, RMX51A.MAC; Appendix A lists the contents of this file. Refer to a previous section in this chapter for a description of the Initial Task Descriptor (ITD). Each %ITD macro in Figure 7-2 has the following seven parameters:

1. Beginning location of the task code. (Note that this location cannot be declared external to this module.)
2. Stack size. Must be at least four bytes.
3. Function ID for the task.
4. Register bank specification for the task. (Refer to the ITD description in this chapter.)
5. Task priority (1 - 4).
6. Interrupts to be assigned to the task. (Refer to the ITD description in this chapter.)
7. The location of the next task descriptor in the chain (if any).

The first task descriptor in Figure 7-2 has the following characteristics:

1. Entry point is "Taskl_Entry."
2. Stack size is eight.
3. Function ID is two.
4. Requests any available register bank. (Since it is the first ITD, it will be assigned "register bank 0.")
5. Priority is four. Four is the highest priority.
6. Interrupts assigned to it are "internal serial port 1", "external request 1", and "external request 0."
7. Next task descriptor specified is ITD2.

The second task descriptor's distinguishing feature is that it requests "register bank 3" and has no interrupts assigned to it.

The third task descriptor's distinguishing feature is that it specifies a one-byte displacement into its own ITD as the next ITD. This specification causes the next task pointer to point to an invalid bit pattern which, in turn, causes the task initialization process to terminate. Refer to the section in this chapter that describes the ITD. This task also has "timer 1" associated with it.

7.2.2 iRMX™ 51 CONFIGURATION EXAMPLE (MODULE 2)

Figure 7-3 lists the contents of "module 2" of the iRMX 51 configuration example. It contains the code for the second iRMX 51 task. This module requires only two INCLUDE files which declare the system externals and literals: RMX51A.EXT and RMX51A.LIT.

```
$INCLUDE(:fx:rmx51a.ext)
$INCLUDE(:fx:rmx51a.lit)

    CSEG

PUBLIC Task1

Task1:
    MOV     A,#011B             ; wait for an interrupt or message
    MOV     B,#20H              ; with timeout
    CALL    RQWAIT
    •
    •                           ; performs interrupt processing
    •

JMP Task1
```

Figure 7-3. iRMX™ 51 Executive Configuration Example (Module 2)

7.2.3 iRMX™ 51 CONFIGURATION EXAMPLE (MODULE 3)

Figure 7-4 lists the contents of "module 3" of the iRMX 51 configuration example. It contains the code for the third iRMX 51 task. Since this task is coded in PL/M 51, it must include RMX51P.EXT and RMX51P.LIT. These files contain the PL/M51 declarations of the system externals and the literal declarations.

```
$registerbank(3)
module$3: DO;

$INCLUDE(:fx:rmx51p.ext)
$INCLUDE(:fx:rmx51p.lit)

Task2:  PROCEDURE PUBLIC;
    DECLARE message$ptr WORD
    acc = 10H;
CALL RQ$SET$INTERVAL;
DO WHILE 1;
    acc = 101B;                 /*wait for time interval or message*/
    b = 0FFH;
    message$ptr = RQ$WAIT;
    status = acc;
            •
            •                   /*performs interrupt processing*/
            •
```

Figure 7-4. iRMX™ 51 Executive Configuration Example (Module 3)

```
    END /*DO WHILE*/
END Task2;

END module$3;
```

Figure 7-4. iRMX™ 51 Executive Configuration Example (Module 3) (continued)

PL/M 51 also requires a specific register bank since the compiled code is always dependent upon the register bank. The code in Figure 7-4 uses "$registerbank(3)" to specify register 3 for this task. Therefore, the task descriptor for this task (in Figure 7-2) specifically requests register bank 3.

7.2.4 USING RMX51A.CFG TO CONFIGURE THE EXAMPLE

Figure 7-5 lists the contents of the iRMX 51 configuration file, RMX51A.CFG. This file is included in the example in Module 1 (Figure 7-2). It contains the rest of the information for iRMX 51 system configuration.

NOTE

The default for "RQSYSBUFSIZE" is 16; it has been changed to 20 for this example.

```
;
;   iRMX™ 51 Configuration File
;

PUBLIC  RQFIRSTITD, RQCLOCKPRIORITY, RQCLOCKTICK
PUBLIC  RQPOOLTOP, RQSYSBUFSIZE

RQFIRSTITD          EQU         ITD1        ;address of first task descriptor
RQCLOCKPRIORITY     EQU         5           ;priority of system clock
RQCLOCKTICK         EQU         -1000       ;(negative of) clock cycles/tick
RQPOOLTOP           EQU         191         ;address of top of free space pool
RQSYSBUFSIZE        EQU         20          ;free space buffer size
```

Figure 7-5. RMX51A.CFG for Configuration Example

The file shown in Figure 7-5 specifies the following information:

1. Location of the first ITD.
2. Relative priority of the system clock (1-5). This system clock is updated only when a task with a lower priority is running. Therefore, if the priority is five, the system clock preempts any task which is running. (The maximum task priority is four.)
3. Duration of one unit of time on the system clock (in microseconds, expressed as a negative number).
4. Address of the highest byte of on-chip memory. This address is used by the system as the top of the free-space pool. The value for an 8051 component is 127 and the value for an 8044 component is 191.
5. Size in bytes of the system buffers. During initialization, the system creates as many buffers of this size for which it has on-chip space. The minimum buffer size is four bytes.

The unmodified contents of this file is listed in Appendix A.

7.2.5 iRMX™ 51 LINKING EXAMPLE

Figure 7-6 lists the command for linking the object modules into a single load module.

```
RL51 :fx:<module 1 name>.obj, &
     :fx:<module 2 name>.obj, &
     :fx:<module 3 name>.obj, &
     :fx:rmx51.lib(rlvec), &
     :fx:rmx51.lib, &
     :fx:rmx51i.lib, &
     :fx:plm51.lib, &
TO   :fx:<output module name> STACK(rmx51_stack)
```

Figure 7-6. Command for Linking Object Modules for iRMX™ 51 Configuration Example

Note that the command in Figure 7-6 includes the file, PLM51.LIB. This file must be included if a PL/M 51-object module is present in the system. Also notice that the "STACK(rmx51_stack)" option places the user object modules in the correct area.

You can also link and locate modules without reconfiguring the iRMX 51 system. Refer to the next section for information about linking user tasks separately from the iRMX 51 Executive.

CAUTION

The linker issues a "WARNING 5: CODE SPACE MEMORY OVERLAP" message which you should ignore.

7.3 CONFIGURING THE DCM CONTROLLER FIRMWARE

You do not have to configure the actual DCM controller firmware. If you want to add additional iRMX 51 user tasks to the DCM controller firmware, you must configure the tasks in much the same manner as described in the previous section. You must also link and locate the tasks so that they can be used with the DCM controller firmware. This requires that the ITD be located at FFF0H as shown in the following example.

The following sections use an example to describe both configuring your DCM tasks and linking and locating them for use with the DCM firmware. The configuration example consists of a DCM system with two user tasks which are contained in one source modules. This source module contains two ASM51 tasks.

7.3.1 AN EXAMPLE CONFIGURATION OF TWO DCM TASKS

Figure 7-7 contains the ITDs and the code for two DCM tasks. This example assumes that the module will be placed on a 4K ROM component with a beginning address of F000H. Since ITD1 is ORGed at 0FF0H (relative to the beginning of ROM) the system locates it absolutely at FFF0H; the DCM firmware can then find the ITD and initialize it.

```
$INCLUDE(:fx:rmx51a.ext)
$INCLUDE(:fx:rmx51a.lit)
$INCLUDE(:fx:rmx51a.mac)

    CSEG
    ORG 0

Task1

  •
  •       ;performs message processing, etc
  •

JMP Task1

Task2

  •
  •       ;Task2 code
  •

JMP Task2
```

Figure 7-7. Example Code for Two DCM Tasks

```
ITD2:
%ITD(Task2, 8, 055, 000B, 2, 00000B, -16)

    ORG  4096-16

ITD1:
%ITD(Task1, 8, 023, 000B, 3, 10000B, ITD2)

END
```

Figure 7-7. Example Code for Two DCM Tasks (Continued)

7.3.2 AN EXAMPLE OF LINKING YOUR USER TASKS FOR USE WITH THE DCM CONTROLLER FIRMWARE

Figure 7-8 lists an example of how to link user tasks for use with the DCM controller firmware.

```
RL51    :fx:<module name>.obj, &
        :fx:RMX51I.LIB, &
    TO  :fx:<output module name>
```

Figure 7-8. Example Command for Linking User Tasks to be Used with the DCM Controller Firmware

CHAPTER 8
REMOTE ACCESS AND CONTROL (RAC) INTERFACE

This chapter describes the Remote Access and Control (RAC) interface. The RAC interface is provided by a preconfigured task that resides on all DCM nodes. (This chapter refers to the task as the "RAC task", but the DCM system knows it as "Task 0.") The RAC task can perform a specific set of services which are described in this chapter.

The RAC task can understand and respond to iRMX 51 order messages. Because the RAC task can understand iRMX 51 order messages, you can access I/O and memory locations, or you can invoke iRMX 51 system calls on remote devices of the DCM system without having to create tasks on these remote nodes.

8.1 RAC SERVICES

The RAC task is part of the DCM controller firmware. As explained in Chapter 2, the DCM controller firmware is available in two different forms. The RAC task allows you to invoke some iRMX 51 system calls; it also allows you to access I/O and memory on remote nodes by sending iRMX 51 messages.

You can invoke the RAC task's services by sending iRMX 51 order messages which include RAC parameters. You can send order messages from your tasks to the RAC task either on the local node, or on remote nodes. Figure 8-1 illustrates how to send an order message from a user task to a RAC task. The task can ask the RAC task to monitor an I/O point or to control/maintain a process. The user task in Figure 8-1 can reside either on the same node as the RAC task or on a different node.

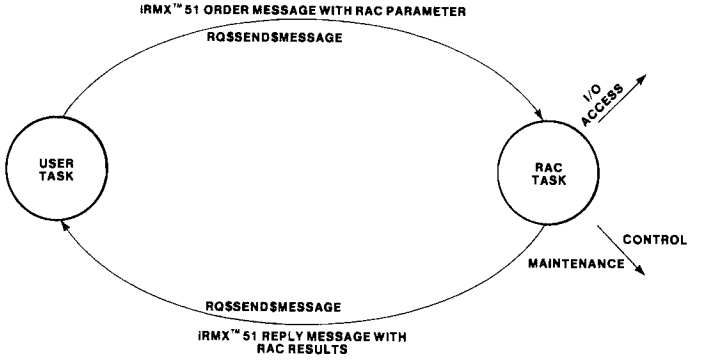


Figure 8-1. Sending a Message from a User Task to the RAC Task

Refer to Chapter 6 for more information about using RQ$SEND$MESSAGE and RQ$WAIT to send messages to the RAC task.

You can invoke two types of RAC services: data access services and task control services. The data access class of services allow you to read and write data to memory and I/O areas and to perform logical functions (AND, OR, etc.). The task control services allow you to execute some iRMX 51 system calls on a remote node.

8.1.1 DATA ACCESS SERVICES

As stated previously, data access services allow you to read and write data to different segments of memory in the 8051 component. The DCM controller firmware uses the external memory segment, 0FF00H-0FFFFH, for I/O functions. You can control how data is written to I/O areas by the following RAC functions: EXT_AND_IO, EXT_OR_IO, EXT_XOR_IO, EXT_READ_IO, EXT_WRITE_IO, and EXT_UPDATE_IO.

In addition, the DCM controller firmware can use the entire external memory segment (00000H-0FFFFH) for other memory functions. These memory functions include the following RAC commands: EXT_UPLOAD and EXT_DNLOAD. Figure 8-2 illustrates how the RAC commands access external memory.

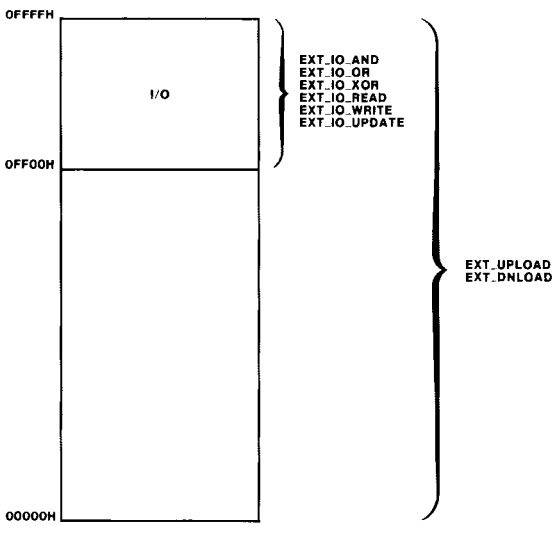


Figure 8-2. How the RAC Commands Access External Data Memory

Refer to the individual RAC service descriptions for more information about how these commands work.

8.1.2 TASK CONTROL SERVICES

As stated previously, task control services allow you to use appropriate iRMX 51 system calls on a remote node or it can allow you to control the remote node, itself. The RAC services pass the parameters for the iRMX 51 system calls as data. The task control functions also include the capability of resetting the remote node's software and turning the RAC data access services on or off.

8.2 RAC MESSAGE FORMAT

This section describes the specific types of values that you must use for RAC message parameters. Refer to Chapter 5 of this manual for more information about message passing and the iRMX 51 message parameters. The format for RAC messages (and iRMX 51 messages) is shown in Figure 8-3. This structure is the same as the BITBUS message format. Refer to the BITBUS section of the DISTRIBUTED CONTROL MODULES DATA BOOK for more information.

```
RAC_Message:        Link                    WORD,
                    Message_length          BYTE,
                    Message_type            BIT,
                    Src_ext                 BIT,
                    Dest_ext                BIT,
                    Trk                     BIT,
                    Reserved                BIT(4)
                    Node_address            BYTE,
                    Source_task_id          BIT(4)
                    Destination_task_id     BIT(4)=0
                    Command/response        BYTE,
                    RAC parameters          (N)BYTES;
```

Figure 8-3. RAC Message Structure

The parameters in Figure 8-3 (as they apply to the RAC task) are as follows:

Link	Reserved. Refer to Chapter 5 for more information about this parameter.
Message_length	The number of bytes in the RAC order or reply. Refer to the RAC services in this chapter for specific values.

Message_type	A bit which indicates the type of message. The bit is clear (0) when the message is a RAC order. The bit is set (1) when a reply is received from the RAC task.
Src_ext	A bit which indicates whether the task generating the message is located on an extension node, or not. The bit is set (1) when the message originates from a task located on an extension node. Otherwise, the bit is clear (0).
Dest_ext	A bit value that indicates whether the task which receives an order message resides on an extension (=1) or on a device (0). Extensions are described in Chapters 9 and 12 of this manual.
Trk	This bit is always set to zero (0).
Node_address	This parameter has different values depending upon where the message is delivered. For messages traveling over the parallel bus only, the parameter is "0FFH." For order messages traveling from a master device (or its extension) to a slave device (or its extension), the parameter is the BITBUS node address of the slave device. Refer to Chapters 9 and 12 for more information about extensions. Refer to the BITBUS section of the DISTRIBUTED CONTROL MODULES DATA BOOK for more information about the BITBUS interconnect.
Source_task_id	A value (0 - 7) which identifies the task which sent the RAC order message.
Destination_task_id	The task identifier for the RAC task. This value is zero (0) for the RAC task.
Command/response	If you have generated a RAC order message, this field contains a parameter which selects the specific RAC service for that message. The reply message from the RAC service contains the status code which indicates the results of the service.
RAC parameters	All bytes following the seven bytes of the message header are a fixed structure dependent upon the specific RAC service type for order messages. The reply message for the RAC task has a fixed set of parameters dependent upon the RAC service it performed. Refer to the specific RAC services for more information.

8.3 RAC FUNCTION DICTIONARY

The RAC function dictionary lists (in alphabetical order) the individual RAC functions which make up the RAC services described in this chapter.

Table 8-1. RAC Function Summary

RAC Functions	Hexadecimal Values	Description
CREATE_TASK	01H	Executes the iRMX 51 RQ$CREATE$TASK system call using the task descriptor_ptr of the RAC message. It returns the result in the reply message.
DELETE_TASK	02H	Executes the iRMX 51 RQ$DELETE$TASK system call using the task_id of the RAC message. It returns the result in the reply message.
EXT_DNLOAD	09H	Writes data starting at external memory location.
EXT_IO_AND	0BH	AND's values with external I/O locations and returns a reply message containing the results.
EXT_IO_OR	0AH	OR's values with contents of external I/O locations and returns a reply message containing the results.
EXT_IO_READ	05H	Reads external I/O location and returns the result in the reply message.
EXT_IO_UPDATE	07H	Writes and then reads byte to and from external I/O location.
EXT_IO_WRITE	06H	Writes byte to external I/O location.
EXT_IO_XOR	0CH	XOR's values with contents of external I/O location and returns a reply message containing the results.
EXT_UPLOAD	08H	Reads data starting at external memory location and returns result in reply message.

Table 8-1. RAC Function Summary (continued)

RAC Functions	Hexadecimal Values	Description
GET_FUNCTION_ID	03H	Executes the iRMX 51 RQGETFUNCTION$IDS system call and returns the result in the reply message.
INT_READ	0EH	Reads contents of internal memory location and returns result in reply message.
INT_WRITE	0DH	Writes data to internal memory location.
RAC_FUNCTION_ID	Service Dependent	The function identification value for the RAC service specified.
RAC_PROTECT	04H	Suspends or resumes remote access services and returns the result in reply message.
RAC_TASK_ID	Service Dependent	The identification value for the RAC task.
RESET_STATION	00H	Jumps to code reset address and does not return a reply.

8.4 ORGANIZATION OF THE RAC SERVICE DESCRIPTIONS

The rest of the chapter consists of detailed descriptions of the RAC services in alphabetical order. Each section lists the message structure of the "order" message and the "reply" message. However, since the RAC message structure is similar to the iRMX 51 message structure, this section describes only the parameters which differ from the iRMX 51 parameter descriptions.

You can find the responses to these services in Appendix A.

8.4.1 CREATE TASK

The CREATE TASK service causes the iRMX 51 RQ$CREATE$TASK system call to execute on the addressed device.

8.4.1.2 CREATE TASK Order Message Structure

Figure 8-2 lists the structure for the CREATE TASK order message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	9
Message_type	BIT	= ORDER (0)
Src_ext	BIT	User specified
Dest_ext	BIT	User specified
Trk	BIT	0
Reserved	BIT(4)	Reserved
Node_address	BYTE	User specified
Source_task_id	BIT(4)	User specified
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= CREATE_TASK
Task_descriptor_ptr	WORD	See description in "parameters" section

Figure 8-4. CREATE TASK Order Message

8.4.1.2 CREATE TASK Reply Message Structure

Figure 8-5 lists the structure for the CREATE TASK reply message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	9
Message_type	BIT	= REPLY (1)
Src_ext	BIT	Unchanged
Dest_ext	BIT	Unchanged
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	Unchanged
Source_task_id	BIT(4)	Unchanged
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= See description in "parameters" section
Task_descriptor_ptr	WORD	Unchanged

Figure 8-5. CREATE TASK Reply Message

The RAC-specific parameters in Figures 8-4 and 8-5 are as follows:

Task_descriptor_ptr — The address of a Task_descriptor that you specified in the 8051 code segment on the remote node. You must set the ITD before you request this service.

Command/Response — The service status of either the message delivery or the RAC service call. The status is one of the following values:

- The iRMX 51 system call statuses for RQ$CREATE$TASK.
- One of the communications system statuses when a message delivery fails.

See Appendix A for a list of the error messages.

8.4.2 DELETE TASK

The DELETE TASK service causes the iRMX 51 RQ$DELETE$TASK system call to execute on the addressed device.

8.4.2.1 DELETE TASK Order Message Structure

Figure 8-6 lists the structure for the DELETE TASK order message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	8
Message_type	BIT	= ORDER (0)
Src_ext	BIT	User specified
Dest_ext	BIT	User specified
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	User specified
Source_task_id	BIT(4)	User specified
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= DELETE_TASK
Task_id	BYTE	See description in "parameters" section

Figure 8-6. DELETE TASK Order Message

8.4.2.2 DELETE TASK Reply Message Structure

Figure 8-7 lists the structure for the DELETE TASK reply message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	8
Message_type	BIT	= REPLY (1)
Src_ext	BIT	Unchanged
Dest_ext	BIT	Unchanged
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	Unchanged
Source_task_id	BIT(4)	Unchanged
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= See description in "parameters" section
Task__id	BYTE	Unchanged

Figure 8-7. DELETE TASK Reply Message

The RAC-specific parameters listed in Figures 8-6 and 8-7 are as follows:

Task_id — The identification value for the task you want to delete on the RAC service's device.

Command/Response — The service status of either the message delivery or the RAC service call. The status is one of the following values:

- The iRMX 51 system call statuses for RQ$DELETE$TASK.
- One of the communications system statuses for message delivery.

8.4.3 DOWNLOAD EXTERNAL MEMORY

The DOWNLOAD EXTERNAL MEMORY service causes the RAC task to write the parameters of the message to a block of external RAM that you specify.

8.4.3.1 DOWNLOAD EXTERNAL MEMORY Order Message Structure

Figure 8-8 lists the structure for the DOWNLOAD EXTERNAL MEMORY order message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	See description in "parameters" section
Message_type	BIT	= ORDER (0)
Src_ext	BIT	User specified
Dest_ext	BIT	User specified
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	User specified
Source_task_id	BIT(4)	User specified
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= EXT_DNLOAD
Memory__ptr	WORD	See description in "parameters" section
Memory_byte_1	BYTE	See description in "parameters" section
*Memory_byte_2	BYTE	See description
*Memory_byte_3	BYTE	See description
*Memory_byte_4	BYTE	See description
*Memory_byte_5	BYTE	See description
*Memory_byte_6	BYTE	See description
*Memory_byte_7	BYTE	See description
*Memory_byte_8	BYTE	See description
*Memory_byte_9	BYTE	See description
*Memory_byte_10	BYTE	See description
*Memory_byte_11	BYTE	See description

Figure 8-8. DOWNLOAD EXTERNAL MEMORY Order Message

Note: The asterisk (*) indicates that these parameters are optional

8.4.3.2 DOWNLOAD EXTERNAL MEMORY Reply Message Structure

Figure 8-9 lists the structure for the DOWNLOAD EXTERNAL MEMORY reply message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	See description in "parameters" section
Message_type	BIT	= REPLY (1)
Src_ext	BIT	Unchanged
Dest_ext	BIT	Unchanged
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	Unchanged
Source_task_id	BIT(4)	Unchanged
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= See description in "parameters" section
Memory_ptr	WORD	Unchanged
Memory_byte_1	BYTE	Unchanged
*Memory_byte_2	BYTE	See description
*Memory_byte_3	BYTE	See description
*Memory_byte_4	BYTE	See description
*Memory_byte_5	BYTE	See description
*Memory_byte_6	BYTE	See description
*Memory_byte_7	BYTE	See description
*Memory_byte_8	BYTE	See description
*Memory_byte_9	BYTE	See description
*Memory_byte_10	BYTE	See description
*Memory_byte_11	BYTE	See description

Figure 8-9. DOWNLOAD EXTERNAL MEMORY Reply Message

Note: The asterisk (*) indicates that these parameters are optional

The RAC-specific parameters in Figures 8-8 and 8-9 are as follows:

Message_length — The length of the message depends upon the number of bytes you wish to download. The value cannot exceed 20; the number of bytes cannot exceed 11.

Memory_ptr — A pointer to the external memory location.

Memory_byte_X	The values the RAC task reads at the external memory location referenced by Memory_ptr. (Memory_byte_2 through Memory byte_11 are optional.) The Message_length parameter sets the number of bytes "X" by the following formula: (X = Message length - 9).
Command/Response	The service status of either the message delivery or the RAC service call. The status is one of the communications system statuses for message delivery.

8.4.4 EXTERNAL I/O ACCESS

The EXTERNAL I/O ACCESS service includes a number of I/O functions. The following is a list and a short description of each function:

EXT_IO_READ	Instructs the RAC task to read the contents of the designated external I/O location. It then returns the contents in the reply message.
EXT_IO_WRITE	Instructs the RAC task to write a specific value to a designated external I/O location.
EXT_IO_UPDATE	Instructs the RAC task to write a specific value to a designated external I/O location. It then reads and returns the contents of that location in the reply message.
EXT_IO_AND	Instructs the RAC task to AND a specific value with the contents of a designated external I/O location.
EXT_IO_OR	Instructs the RAC task to OR a specific value with the contents of a designated external I/O location.
EXT_IO_XOR	Instructs the RAC task to XOR a specific value with the contents of a designated external I/O location.

8.4.4.1 EXTERNAL I/O ACCESS Order Message Structure

Figure 8-10 lists the structure for a common EXTERNAL I/O ACCESS order message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	See description in "parameters" section
Message_type	BIT	= ORDER (0)
Src_ext	BIT	User specified
Dest_ext	BIT	User specified
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	User specified
Source_task_id	BIT(4)	User specified
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= See description in "parameters" section

Figure 8-10. Common EXTERNAL I/O ACCESS Order Message

I/O_ptr_1	BYTE	See description in "parameters" section
I/O_byte_1	BYTE	See description in "parameters" section
*I/O_ptr_2	BYTE	See description
*I/O_byte_2	BYTE	See description
*I/O_ptr_3	BYTE	See description
*I/O_byte_3	BYTE	See description
*I/O_ptr_4	BYTE	See description
*I/O_byte_4	BYTE	See description
*I/O_ptr_5	BYTE	See description
*I/O_byte_5	BYTE	See description
*I/O_ptr_6	BYTE	See description
*I/O_byte_6	BYTE	See description

Figure 8-10. Common EXTERNAL I/O ACCESS Order Message (continued)

Note: The asterisk (*) indicates that these parameters are optional

8.4.4.2 EXTERNAL I/O ACCESS Reply Message Structure

Figure 8-11 lists the structure for a common EXTERNAL I/O ACCESS reply message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	See description in "parameters" section
Message_type	BIT	= REPLY (1)
Src_ext	BIT	Unchanged
Dest_ext	BIT	Unchanged
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	Unchanged
Source_task_id	BIT(4)	Unchanged
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= See description in "parameters" section

Figure 8-11. Common EXTERNAL I/O ACCESS Reply Message

I/O_ptr_1	BYTE	See description in "parameters" section
I/O_byte_1	BYTE	See description in "parameters" section
*I/O_ptr_2	BYTE	See description
*I/O_byte_2	BYTE	See description
*I/O_ptr_3	BYTE	See description
*I/O_byte_3	BYTE	See description
*I/O_ptr_4	BYTE	See description
*I/O_byte_4	BYTE	See description
*I/O_ptr_5	BYTE	See description
*I/O_byte_5	BYTE	See description
*I/O_ptr_6	BYTE	See description
*I/O_byte_6	BYTE	See description

Figure 8-11. Common EXTERNAL I/O ACCESS Reply Message (continued)

Note: The asterisk (*) indicates that these parameters are optional

The RAC-specific parameters in Figures 8-10 and 8-11 are as follows:

Message_length (in Figure 8-10 & 8-11)
: The number of bytes in the message. I/O_ptr_2 and I/O_byte_2 through I/O_ptr_6 and I/O_byte_6 are optional. Message length determines whether these parameters are included. The minimum Message_length = 9 and the maximum Message_length = 19.

Command/Response (in Figure 8-10)
: Indicates the type of I/O access. The valid types are as follows:

EXT_IO_READ	EXT_IO_WRITE
EXT_IO_UPDATE	EXT_IO_AND
EXT_IO_OR	EXT_IO_XOR

I/O_ptr_X
: A pointer to the external I/O location. External I/O is memory-mapped in locations OFFOOH to OFFFFH. The I/O_ptr is a byte-offset based from OFFOOH in this space.

I/O_byte_X
: A place holder for the value to be accessed from the external I/O location referenced by I/O_ptr_X. The access can be any of the following functions: EXT_IO READ, EXT_IO_WRITE, EXT_IO_UPDATE, EXT_IO AND, EXT_IO_OR, and EXT_IO_XOR.

Command/Response (in Figure 8-11)	The service status of either the message delivery or the RAC service call. The status is one of the communications system statuses for message delivery.
I/O_byte_X (in Figure 8-11)	The value returned after the appropriate RAC service has been performed on the external I/O location. This value is available after performing the required access, except in the case of EXT_IO_WRITE.

8.4.5 GET FUNCTION IDS

The GET FUNCTION IDS service instructs the RAC task to execute the iRMX 51 RQGETFUNCTION$IDS system call on the addressed device.

8.4.5.1 GET FUNCTION IDS Order Message Structure

Figure 8-12 lists the structure for the GET FUNCTION IDS order message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	15
Message_type	BIT	= ORDER (0)
Src_ext	BIT	User specified
Dest_ext	BIT	User specified
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	User specified
Source_task_id	BIT(4)	User specified
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= GET_FUNCTION_ID

Figure 8-12. GET FUNCTION IDS Order Message

8.4.5.2 GET FUNCTION IDS Reply Message Structure

Figure 8-13 lists the structure for the GET FUNCTION IDS reply message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	15
Message_type	BIT	= REPLY (1)
Src_ext	BIT	Unchanged
Dest_ext	BIT	Unchanged
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	Unchanged
Source_task_id	BIT(4)	Unchanged
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= See description in "parameters" section
Data	BYTE	Reserved

Figure 8-13. GET FUNCTION IDS Reply Message

*Task_0_function_id	BYTE	RAC_FUNCTION_ID
*Task_1_function_id	BYTE	See description
*Task_2_function_id	BYTE	See description
*Task_3_function_id	BYTE	See description
*Task_4_function_id	BYTE	See description
*Task_5_function_id	BYTE	See description
*Task_6_function_id	BYTE	See description
*Task_7_function_id	BYTE	See description

Figure 8-13. GET FUNCTION IDS Reply Message (continued)

The RAC-specific parameters in Figures 8-12 and 8-13 are as follows:

Command/Response

The service status of either the message delivery or the RAC service call. The status is one of the following values:

- The iRMX 51 system call statuses for RQGETFUNCTION$ID.
- One of the communications system statuses for message delivery.

Task_X_function_id

The function_id for the task with a task id of "X" on the RAC service's device. If no task is associated with the id, the result is zero (0). The function_id for the RAC task (Task 0) must be one. Refer to BITBUS section of the DISTRIBUTED CONTROL MODULES DATA BOOK for more information on function ids.

8.4.6 PROTECT RAC

The PROTECT RAC service allows you to protect the RAC task from any type of memory or I/O access. This service also allows you to re-instate the accesses.

8.4.6.1 PROTECT RAC Order Message Structure

Figure 8-14 lists the structure for the PROTECT RAC order message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	8
Message_type	BIT	= ORDER (0)
Src_ext	BIT	User specified
Dest_ext	BIT	User specified
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	User specified
Source_task_id	BIT(4)	User specified
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= RAC_PROTECT
Protect_flag	BYTE	See description in "parameters" section

Figure 8-14. PROTECT RAC Order Message

8.4.6.2 PROTECT RAC Reply Message Structure

Figure 8-15 lists the structure for the PROTECT RAC reply message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	8
Message_type	BIT	= REPLY (1)
Src_ext	BIT	Unchanged
Dest_ext	BIT	Unchanged
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	Unchanged
Source_task_id	BIT(4)	Unchanged
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= See description in "parameters" section
Protect_flag	BYTE	See description in "parameters" section

Figure 8-15. PROTECT RAC Reply Message

The RAC-specific parameters in Figures 8-14 and 8-15 are as follows:

Protect_flag	When this flag is set (1), the DCM system ignores RAC services which access memory or I/O functions. When the flag is zero (0), the RAC access services can function.
Command/Response	The service status of either the message delivery or the RAC service call. The status is one of the communications system statuses for message delivery.

8.4.7 READ INTERNAL MEMORY

The READ INTERNAL MEMORY service instructs the RAC task to read the contents of up to six specific on-chip (internal data segment) memory locations.

8.4.7.1 READ INTERNAL MEMORY Order Message Structure

Figure 8-16 lists the structure for the READ INTERNAL MEMORY order message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	See description in "parameters" section
Message_type	BIT	= ORDER (0)
Src_ext	BIT	User specified
Dest_ext	BIT	User specified
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	User specified
Source_task_id	BIT(4)	User specified
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= INT_READ
Memory__ptr_1	BYTE	See description in "parameters" section
Memory_byte_1	BYTE	See description in "parameters" section
*Memory_ptr_2	BYTE	See description
*Memory_byte_2	BYTE	See description
*Memory_ptr_3	BYTE	See description
*Memory_byte_3	BYTE	See description
*Memory_ptr_4	BYTE	See description
*Memory_byte_4	BYTE	See description
*Memory_ptr_5	BYTE	See description
*Memory_byte_5	BYTE	See description
*Memory_ptr_6	BYTE	See description
*Memory_byte_6	BYTE	See description

Figure 8-16. READ INTERNAL MEMORY Order Message

Note: The asterisk (*) indicates that these parameters are optional

8.4.7.2 READ INTERNAL MEMORY Reply Message Structure

Figure 8-17 lists the structure for the READ INTERNAL MEMORY reply message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	See description in "parameters" section
Message_type	BIT	= REPLY (1)
Src_ext	BIT	Unchanged
Dest_ext	BIT	Unchanged
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	Unchanged
Source_task_id	BIT(4)	Unchanged
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= See description in "parameters" section
Memory__ptr_1	BYTE	Unchanged
Memory_byte_1	BYTE	See description
*Memory_ptr_2	BYTE	Unchanged
*Memory_byte_2	BYTE	See description
*Memory_ptr_3	BYTE	Unchanged
*Memory_byte_3	BYTE	See description
*Memory_ptr_4	BYTE	Unchanged
*Memory_byte_4	BYTE	See description
*Memory_ptr_5	BYTE	Unchanged
*Memory_byte_5	BYTE	See description
*Memory_ptr_6	BYTE	Unchanged
*Memory_byte_6	BYTE	See description

Figure 8-17. READ INTERNAL MEMORY Reply Message

Note: The asterisk (*) indicates that these parameters are optional

The RAC-specific parameters in Figures 8-16 and 8-17 are as follows:

Memory_ptr_X	A pointer to an internal memory location.
Message_length	The number of bytes in the message. Memory_ptr_2 and Memory_byte_2 through Memory_ptr_6 and Memory_byte_6 are optional. Message_length determines whether these parameters are included. The minimum Message_length = 9 and the maximum Message_length = 19.

Command/Response	The service status of either the message delivery or the RAC service call. The status is one of the communications system statuses for message delivery. Appendix A contains the (hexadecimal) responses.
Memory_byte_X	The value read from the internal memory location referenced by memory_ptr_X.

8.4.8 RESET DEVICE

The RESET DEVICE service instructs the RAC task to jump to the software reset location of the device, thus resetting the software functions.

8.4.8.1 RESET DEVICE Order Message Structure

Figure 8-18 lists the structure for the RESET DEVICE order message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	7
Message_type	BIT	= ORDER (0)
Src_ext	BIT	User specified
Dest_ext	BIT	User specified
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	User specified
Source_task_id	BIT(4)	User specified
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= RESET_STATION

Figure 8-18. RESET DEVICE Order Message

8.4.8.2 RESET DEVICE Reply Message Structure

The RAC task does not generate a RESET DEVICE reply message except for a message protocol of data link protocol error.

8.4.9 UPLOAD EXTERNAL MEMORY

The UPLOAD EXTERNAL MEMORY service instructs the RAC task to read the contents of a block of external RAM.

8.4.9.1 UPLOAD EXTERNAL MEMORY Order Message Structure

Figure 8-19 lists the structure for the UPLOAD EXTERNAL MEMORY order message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	See the description in the "parameters" section.
Message_type	BIT	= ORDER (0)
Src_ext	BIT	User specified
Dest_ext	BIT	User specified
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	User specified
Source_task_id	BIT(4)	User specified
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= EXT_UPLOAD
Memory_ptr	WORD	See the description in the "parameters" section.
Memory_byte_1	BYTE	See description
*Memory_byte_2	BYTE	See description
*Memory_byte_3	BYTE	See description
*Memory_byte_4	BYTE	See description
*Memory_byte_5	BYTE	See description
*Memory_byte_6	BYTE	See description
*Memory_byte_7	BYTE	See description
*Memory_byte_8	BYTE	See description
*Memory_byte_9	BYTE	See description
*Memory_byte_10	BYTE	See description
*Memory_byte_11	BYTE	See description

Figure 8-19. UPLOAD EXTERNAL MEMORY Order Message

Note: The asterisk (*) indicates that these parameters are optional

8.4.9.2 UPLOAD EXTERNAL MEMORY Reply Message Structure

Figure 8-20 lists the structure for the UPLOAD EXTERNAL MEMORY reply message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	See the description in the "parameters" section.
Message_type	BIT	= REPLY (1)
Src_ext	BIT	Unchanged
Dest_ext	BIT	Unchanged
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	Unchanged
Source_task_id	BIT(4)	Unchanged
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= See the description in the "parameters" section
Memory_ptr	WORD	Unchanged
Memory_byte_1	BYTE	See description
*Memory_byte_2	BYTE	See description
*Memory_byte_3	BYTE	See description
*Memory_byte_4	BYTE	See description
*Memory_byte_5	BYTE	See description
*Memory_byte_6	BYTE	See description
*Memory_byte_7	BYTE	See description
*Memory_byte_8	BYTE	See description
*Memory_byte_9	BYTE	See description
*Memory_byte_10	BYTE	See description
*Memory_byte_11	BYTE	See description

Figure 8-20. UPLOAD EXTERNAL MEMORY Reply Message

Note: The asterisk (*) indicates that these parameters are optional

The RAC-specific parameters in Figures 8-19 and 8-20 are as follows:

Message_length	The number of bytes to be uploaded. This value cannot exceed 20; the number of bytes to be uploaded cannot exceed 11.
Memory_ptr	A pointer to the external memory location from which the upload begins.
Command/Response	The service status of either the message delivery or the RAC service call. The status is one of the communications system statuses for message delivery.
Memory_byte_X	The value read starting at the external memory location referenced by Memory_ptr. The Message_length parameter sets the number of bytes "X" by the following formula: (X = Message_length - 9).

8.4.10 WRITE INTERNAL MEMORY

The WRITE INTERNAL MEMORY service instructs the RAC task to write up to six values to designated on-chip (internal data segment) RAM locations.

8.4.10.1 WRITE INTERNAL MEMORY Order Message Structure

Figure 8-21 lists the structure for the WRITE INTERNAL MEMORY order message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	See description in the "parameters" section.
Message_type	BIT	= ORDER (0)
Src_ext	BIT	User specified
Dest_ext	BIT	User specified
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	User specified
Source_task_id	BIT(4)	User specified
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= INT_WRITE
Memory__ptr_1	BYTE	See description in "parameters" section
Memory_byte_1	BYTE	See description in "parameters" section
*Memory_ptr_2	BYTE	See description
*Memory_byte_2	BYTE	See description
*Memory_ptr_3	BYTE	See description
*Memory_byte_3	BYTE	See description
*Memory_ptr_4	BYTE	See description
*Memory_byte_4	BYTE	See description
*Memory_ptr_5	BYTE	See description
*Memory_byte_5	BYTE	See description
*Memory_ptr_6	BYTE	See description
*Memory_byte_6	BYTE	See description

Figure 8-21. WRITE INTERNAL MEMORY Order Message

Note: The asterisk (*) indicates that these parameters are optional

8.4.10.2 WRITE INTERNAL MEMORY Reply Message Structure

Figure 8-22 lists the structure for the WRITE INTERNAL MEMORY reply message along with the values of parameters.

Link	WORD	0
Message_length	BYTE	See description in "parameters" section
Message_type	BIT	= REPLY (1)
Src_ext	BIT	Unchanged
Dest_ext	BIT	Unchanged
Trk	BIT	0
Reserved	BIT(4)	Completes BYTE
Node_address	BYTE	Unchanged
Source_task_id	BIT(4)	Unchanged
Destination_task_id	BIT(4)	= RAC_TASK_ID (0)
Command/Response	BYTE	= See description in "parameters" section
Memory__ptr_1	BYTE	Unchanged
Memory_byte_1	BYTE	See description
*Memory_ptr_2	BYTE	Unchanged
*Memory_byte_2	BYTE	Unchanged
*Memory_ptr_3	BYTE	Unchanged
*Memory_byte_3	BYTE	Unchanged
*Memory_ptr_4	BYTE	Unchanged
*Memory_byte_4	BYTE	Unchanged
*Memory_ptr_5	BYTE	Unchanged
*Memory_byte_5	BYTE	Unchanged
*Memory_ptr_6	BYTE	Unchanged
*Memory_byte_6	BYTE	Unchanged

Figure 8-22. WRITE INTERNAL MEMORY Reply Message

Note: The asterisk (*) indicates that these parameters are optional

The RAC-specific parameters in Figures 8-21 and 8-22 are as follows:

Memory_ptr_X	A pointer to an internal memory location.
Message_length	The number of bytes in the message. Memory_ptr_2 and Memory_byte_2 through Memory_ptr_6 and Memory_byte_6 are optional. Message_length determines whether these parameters are included. The minimum Message_length = 9 and the maximum Message_length = 19.

Memory_byte_X	The value to be written to the internal memory location referenced by Memory_ptr X. This value remains unchanged in the reply message.
Command/Response	The service status of either the message delivery or the RAC service call. The status is one of the communications system statuses for message delivery.

8.5 CONFIGURING THE RAC INTERFACE

The RAC task commands are defined in the file DCM44A. LIT (for ASM 51 applications) and in the file, DCM44P.LIT (for PL/M-51 applications). The contents of these files are listed in Appendix C.

You can include one of these files in the your task code when you configure the DCM system. Refer to Chapter 7 for information about configuring the DCM controller firmware; it contains configuration information about both the DCM controller firmware and the iRMX 51 Executive on the iRMX 51 Support Package diskette.

***

CHAPTER 9
iRMX™ 510 OPERATING SYSTEM HANDLERS

This chapter describes the iRMX 510 Operating System handlers which are supplied as part of the iRMX 510 Support Package. It explains the purpose of the iRMX 510 Operating System handlers, the equipment you need to use the handlers, and how to interface between the iRMX 51 Executive and other Intel operating systems.

9.1 PURPOSE OF THE iRMX™ 510 OPERATING SYSTEM HANDLERS

The iRMX 510 Operating System handlers allow the following Intel operating systems to communicate with the DCM system:

- iRMX 86 and iRMX 286R Operating Systems
- iRMX 88 Executive
- iPDS/ISIS-II Operating System on the iPDS Personal Development System

In order to use the iRMX 510 Support Package you must plug an iSBX 344 BITBUS Controller MULTIMODULE board into the iSBX connection on your processor board. (Refer to Chapter 13 for more information about the iSBX 344 board.)

Table 9-1 lists the operating systems and some of the possible the underlying processor boards that can use the iRMX 510 Operating System handlers to communicate with a DCM system:

Table 9-1. iRMX™ 510 Interface to Intel Operating Systems and Boards

Processor Board	Operating System
iSBC 86/05 iSBC 86/14 iSBC 86/30 iSBC 186/03 iSBC 186/51 iSBC 88/25 iSBC 88/45	iRMX 86 or iRMX 88
iSBC 88/40	iRMX 88
iSBC 286/10	iRMX 286R
iPDS Boards	ISIS-II on the iPDS system

The iRMX 510 Operating System handler allows you to exchange information between an Intel operating system and a DCM system over the iSBX connection. Figure 9-1 illustrates how the iRMX 510 Operating System handler allows DCM and iRMX 86 tasks to exchange information.

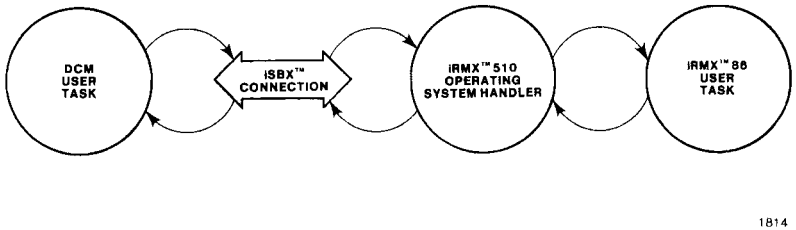


Figure 9-1. Exchanging Information Between a DCM System and an iRMX™ 86 Operating System

9.2 NECESSARY EQUIPMENT

In addition to the iRMX 510 Support Package, you need the following equipment to use the iRMX 510 Operating System handlers:

- An iSBX 344 BITBUS Controller MULTIMODULE board.
- An Intel iSBC board with a iSBX MULTIMODULE connection.
- An Intel operating system (as previously specified).

9.3 PHYSICAL HARDWARE INTERFACES BETWEEN INTEL OPERATING SYSTEMS AND A DCM SYSTEM

In order to use the iRMX 510 Operating System handlers to communicate with Intel Operating Systems, you must insert the iSBX 344 board into the iSBX connector on the processor board which controls the other Intel Operating System. Then, connect the BITBUS bus to the BITBUS connection on the iSBX 344 board. These connections (plus the iRMX 510 handler) allows the DCM system and other Intel operating systems to communicate.

Figure 9-2 illustrates the physical connection between an iRMX 86 system and DCM system. Note that the iRMX 86 system in Figure 9-2 can act as either a slave extension or a master extension.

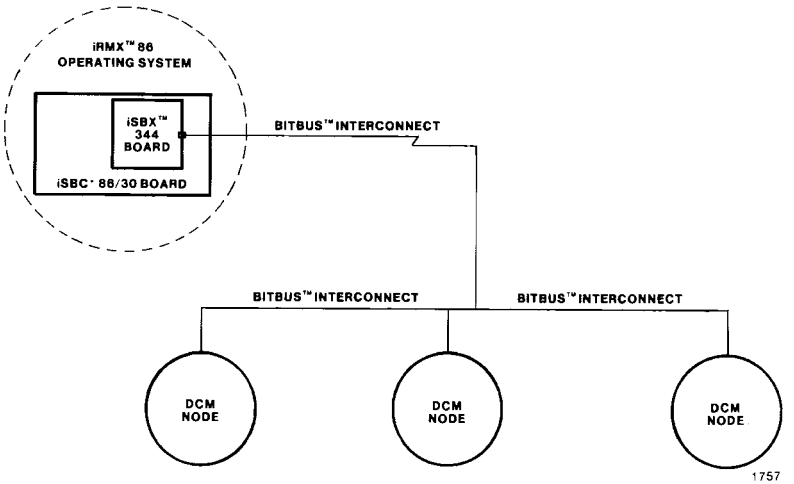


Figure 9-2. Physical Connections between an iRMX™ 86 System and a DCM system

9.4 LOGICAL (SOFTWARE) INTERFACES BETWEEN INTEL OPERATING SYSTEMS AND A DCM SYSTEM

This section describes how to use the iRMX 510 Operating System handlers to allow Intel Operating systems and DCM systems to communicate. It explains how to send and receive messages between the different operating systems and the DCM system. The following subsections are organized so that you need to read only the parts which concern your particular system.

9.4.1 iRMX™ 86 AND iRMX™ 286R INTERFACE

The iRMX 510 handler for the iRMX 86 and iRMX 286R Operating System allows users to take advantage of the services a DCM system offers. This handler passes iRMX 51 messages to and from the iSBX 344 parallel interface.

To transmit a message through the parallel interface, you must use RQ$SEND$MESSAGE to send an iRMX 86 segment (whose data area consists of an iRMX 51 message) to the mailbox associated with the transmit service. You must also use RQ$RECEIVE$MESSAGE to specify the response mailbox to which you want the iRMX 510 handler to return the message.

To receive a message through the parallel interface, you must use RQ$SEND$MESSAGE to send an iRMX 86 segment (whose data area is used as a buffer to store the iRMX 51 message) to the mailbox associated with the receive service. You must also use the RQ$RECEIVE$MESSAGE receive the reply message from the iRMX 510 handler.

The following sections describe how to:

- Locate the "receive" and "transmit" mailboxes.
- Interact with the iRMX 510 handler.

9.4.1.1 Locating the Receive and Transmit Mailboxes

When the iRMX 510 handler is initialized, it creates the "receive" and "transmit" mailboxes and stores their symbolic names, along with their mailbox tokens, in the root job's directory. You can locate these mailboxes by using the iRMX 86/286R RQ$LOOKUP system call at run-time. The procedure for locating the receive and transmit mailboxes is as shown in Figure 9-3. You must call this procedure before sending messages to the iRMX 510 handler.

```
root_job_token    =  RQ$GET$TASK$TOKENS  (root_job,
                                          usr_status_ptr);

receive_mailbox   =  RQ$LOOKUP$OBJECT    (root_job_token,
                                          receive_mailbox_name_ptr,
                                          time_limit,
                                          usr_status_ptr);

transmit_mailbox  =  RQ$LOOKUP$OBJECT    (root_job_token,
                                          transmit_mailbox_name_ptr,
                                          time_limit,
                                          usr_status_ptr);
```

Figure 9-3. Procedure for Locating the Receive and Transmit Mailboxes

NOTE

Refer to the iRMX 86 NUCLEUS REFERENCE MANUAL for more information about how to use the iRMX 86/286R system calls in Figure 9-3.

The definitions for the parameters in Figure 9-3 are as follows:

root_job	A byte value indicating the root job's directory. This value is a constant (3).
except$ptr	A pointer to the variable in which the system places the results of the call.
root_job_token	The token for the root job.
receive_mailbox_name_ptr	A pointer to the string which contains the iRMX 86/286R name for the iRMX 510 handler's receive mailbox. The value that goes here is as follows: @(12, 'DCM_RECV_MBOX').
transmit_mailbox_name_ptr	A pointer to the string which contains the iRMX 86/286R name for the iRMX 510 handler's transmit mailbox. The value that goes here is as follows: @(12, 'DCM_XMIT_MBOX')
time_limit	The word value for the length of time the system waits for the name to be posted in the root job's directory. A zero (0) means "don't wait"; a value of 0FFFFH means "wait forever."
receive_mailbox	The token to the iRMX 510 handler's receive mailbox.
transmit_mailbox	The token to the iRMX 510 handler's transmit mailbox.

9.4.1.2 Interacting with the iRMX™ 510 Operating System Handler for the iRMX™ 86 Operating System

This section explains how to use the iRMX 510 handler to exchange information between an iRMX 86/286R system and a DCM system. A task running on an iRMX 86/286R system can send messages to and receive messages from a task running on the DCM system. This section describes how to interact with the iRMX 510 handler so that the two systems can communicate.

9.4.1.2.1 SENDING MESSAGES TO THE iRMX™ 510 HANDLER. You can transmit information between tasks running on a DCM system and tasks running on an iRMX 86/286R system by exchanging messages through the iRMX 510 Operating System handler. Figure 9-4 shows the structure of the iRMX 51 message. It is the same structure described in Chapter 5 of this manual

```
iRMX_51_Message:    Link                Word,
                    Message_length      Byte,
                    Message_type        Bit,
                    Src_ext             Bit,
                    Dest_ext            Bit,
                    Trk                 Bit,
                    Reserved            Bit(4),
                    Node_address        Byte,
                    Source_task_id      Bit(4),
                    Destination_task_id Bit(4),
                    Command/response    Byte,
                    Message_information Byte(message_length - 7);
```

Figure 9-4. Sending iRMX™ 51 Messages to the iRMX™ 510 Handler (for iRMX™ 86/286R Tasks)

When sending a message from an iRMX 86/286R task to a task on the DCM system, you must send an iRMX 86 message to the "transmit" mailbox (DCM XMIT_MBX). The message buffer should contain a message with the iRMX 51 message structure and the appropriate fields initialized. The message length shows the length of the message, not the length of the buffer.

You must then wait for a response from the iRMX 510 Operating System handler that acknowledges the message. Your task must wait for this message at the response mailbox you specified when you sent the message. Figure 9-5 illustrates how an iRMX 86 user task sends a message to a DCM user task.

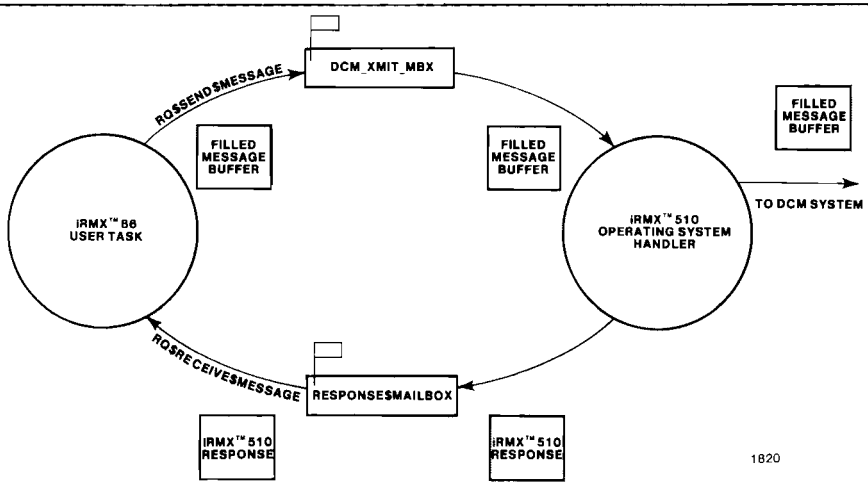


Figure 9-5. An iRMX™ 86 User Task Sending a Message to a User Task on a DCM System

When an iRMX 86/286R task is receiving a message from a task on the DCM system, you should send an iRMX 86 message to the iRMX 510 "receive" mailbox (DCM_RECV_MBX). This message buffer does not have to be initialized, by it should be long enough to accommodate the largest possible message you could receive from the DCM system. (The largest message you could receive from a task running on the DCM system is 20 bytes. The message length on a stand-alone iRMX 51 system is configurable; refer to Chapter 7 for more information about configuring a stand-alone iRMX 51 system.)

You must then wait at the response mailbox (you specified when you sent the message) in order to receive the iRMX 86 message that contains a message from a DCM system task. Figure 9-6 illustrates how an iRMX 86 user task receives a message from a task running on a DCM system.

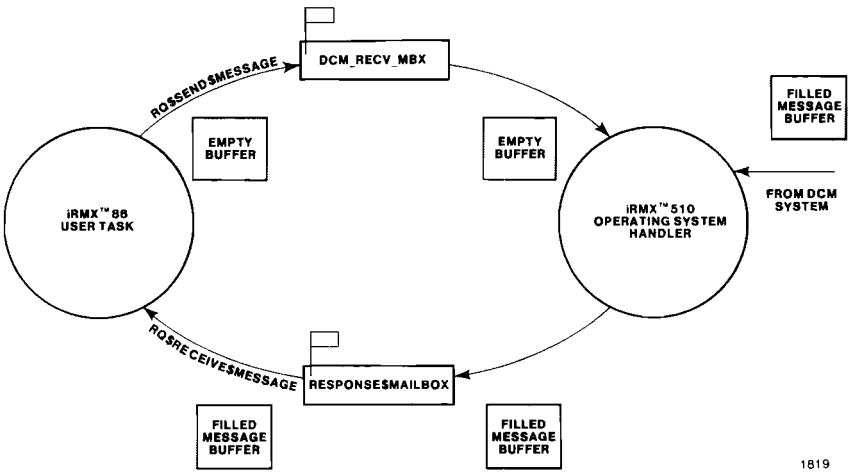


Figure 9-6. An iRMX™ 86 User Task Receiving a Message from a Task Running on a DCM system

9.4.1.2.2 SENDING iRMX™ 86/286R MESSAGES TO THE iRMX™ 510 HANDLER. You can get information from an iRMX 86/286R task to a DCM task by sending iRMX 86 messages to the iRMX 510 handler. You must send an iRMX 86 segment to the iRMX 510 handler; the handler then transmits the information in that segment to the DCM task. The procedure for sending messages to the iRMX 510 handler from an iRMX 86/286R task is shown in Figure 9-7.

```
Call RQ$SEND$MESSAGE        (input_mailbox,
                             handler_message_ptr,
                             response_mailbox,
                             usr_status_ptr);
```

Figure 9-7. Sending A Message (Or Posting A Buffer) To The iRMX™ 510 Handler From An iRMX™ 86/286R Task

NOTE

Refer to the iRMX 86 NUCLEUS REFERENCE MANUAL for more information about how to use the iRMX 86/286R system calls in Figure 9-7.

The definitions for the parameters in Figure 9-7 are as follows:

input_mailbox	The token for one of the iRMX 510 handler mailboxes: the receive_mailbox or the transmit_mailbox.
handler_message_ptr	The token for the iRMX 86 message as defined in Figure 9-3.
response_mailbox	The token for the response mailbox to which the system returns the message after it completes the service.
usr_status_ptr	The pointer to the variable in which the system places the status of RQ$SEND$MESSAGE.

9.4.1.2.3 RECEIVING iRMX™ 86/286R MESSAGES FROM THE iRMX™ 510 HANDLER. DCM tasks receive iRMX 86/286R messages from the iRMX 510 handler. The procedure for receiving iRMX 86/286R messages from the iRMX 510 handler is shown in Figure 9-8.

```
handler_message_ptr = RQ$RECEIVE$MESSAGE  (response_mailbox,
                                           time_limit,
                                           senders_response_mailbox_ptr,
                                           usr_status_ptr);
```

Figure 9-8. Procedure for iRMX™ 86/286R Systems to Receive Messages for a DCM System

The definitions for the parameters in Figure 9-8 are as follows:

response_mailbox	The token for the user response mailbox that was set in the RQ$SEND$MESSAGE system call.
time_limit	The word value for the amount of time the system waits for the response to arrive.
senders_response_ mailbox_ptr	The pointer to a word in which the system places the response mailbox parameter that the iRMX 510 handler sets. The handler sets this parameter when it makes the RQ$SEND$MESSAGE system call to the user's response_mailbox. The value must be zero.
handler_message_ptr	The pointer to the iRMX 86 message as defined in Figure 9-4.

9.4.2 iRMX™ 88 INTERFACE

The iRMX 510 handler for the iRMX 88 Executive allows users to take advantage of the services a DCM system offers. This handler passes iRMX 51 messages to and from the iSBX 344 parallel interface.

To transmit a message through the parallel interface, you must send an use RQ$SEND$MESSAGE to send an iRMX 88 message (whose data area consists of an iRMX 51 message) to the exchange associated with the transmit service. You must also use RQ$RECEIVE$MESSAGE to specify the response exchange to which you want the iRMX 510 handler to return the message.

To receive a message through the parallel interface, you must use RQ$SEND$MESSAGE to send an iRMX 88 message (whose data area is used as a buffer to store the iRMX 51 message) to the exchange associated with the receive service. You must also use RQ$RECEIVE$MESSAGE to receive the reply message from the iRMX 510 handler.

The following sections describe how to:

- Locate the "receive" and "transmit" exchanges.
- Interact with the iRMX 510 handler.

9.4.2.1 Locating the Receive and Transmit Exchanges

The iRMX 510 handler creates the receive and transmit exchanges during initialization. You define these exchanges when you configure the iRMX 510 handler into your iRMX 88 system. You can locate these exchanges when you link the iRMX 88 system. Refer to a later section in this chapter for more information on configuring and linking the iRMX 88 Executive with the iRMX 510 handler.

NOTE

The iRMX 510 Operating System handler supports both the large and compact model of computation for the iRMX 88 Executive, but only in megabyte addressing.

9.4.2.2 Interacting with the iRMX™ 510 Operating System Handler for the iRMX™ 88 Executive

This section explains how to use the iRMX 510 handler to exchange information between an iRMX 88 system and a DCM system. A task running on the iRMX 88 system can send and receive messages from a task running on the DCM system. This section describes how to interact with the iRMX 510 handler so that the two systems can communicate.

9.4.2.2.1 SENDING iRMX™ 51 MESSAGES TO THE iRMX™ 510 HANDLER. You can transmit information between tasks running on a DCM system and tasks running on an iRMX 88 system by exchanging messages through the iRMX 510 Operating System handler. Figure 9-9 shows the structure of the iRMX 51 message. It is the same structure described in Chapter 5 of this manual.

```
Link:                       POINTER
Handler_message_length      WORD
Handler_command_type:       BYTE
Handler_home_exchange:      POINTER
Handler_response_exchange   POINTER

iRMX_51_Message:            Link                Word,
                            Message_length      Byte,
                            Message_type        Bit,
                            Src_ext             Bit,
                            Dest_ext            Bit,
                            Trk                 Bit,
                            Reserved            Bit(4),
                            Node_address        Byte,
                            Source_task_id      Bit(4),
                            Destination_task_id Bit(4),
                            Command/response    Byte,
                            Message_information Byte(message_length - 7);
```

Figure 9-9. Sending iRMX™ 51 Messages to the iRMX™ 510 Handler (for iRMX™ 88 Tasks)

The definitions for the parameters in Figure 9-9 are as follows:

Link	An iRMX 88-specific parameter. Refer to the iRMX 88 REFERENCE MANUAL for more information.
Handler_message_length	The number of bytes in the iRMX 88 message. This value also defines the number of bytes for "Handler_command_response," "Handler message_length," and the "iRMX_51_Message" area.
Handler_command_type	Not used.
Handler_home_exchange	Not used.
Handler_response_exchange	The exchange to which the system returns the message, once it has been serviced. You must initialize this exchange before sending the message to one of the iRMX 510 handler's exchanges.
iRMX_51_Message	A message structure defined in Chapter 5 of this manual.

You send and receive messages in the same manner as described previously for interacting with an iRMX 86/286R system. (Refer to Figures 9-5 and 9-6 for more detailed information.) The only difference is that you must use iRMX 88 calls and the exchanges you specified during configuration. Refer to a later section for information about configuring the iRMX 510 Operating System Handlers.

9.4.2.2.2 SENDING iRMX™ 88 MESSAGES TO THE iRMX™ 510 HANDLER. You can get information from an iRMX 88 task to a DCM task by sending iRMX 88 messages to the iRMX 510 handler. The procedure for sending messages to the iRMX 510 handler from an iRMX 88 task is shown in Figure 9-10.

```
Call RQ$SEND                    (input_exchange,
                                handler_message_ptr);
```

Figure 9-10. Sending iRMX™ 88 Messages to the iRMX™ 510 Handler

NOTE

Refer to the iRMX 88 REFERENCE MANUAL for more information about how to use the iRMX 88 call in Figure 9-10.

The definitions of the parameters in Figure 9-10 are as follows:

input_exchange	A pointer to either "DCM_recv_exchange" or "DCM_xmit_exchange."
handler_message_ptr	A pointer to the iRMX 88 message defined in Figure 9-10.

9.4.2.2.3 RECEIVING iRMX™ 88 MESSAGES FROM THE iRMX™ 510 HANDLER. DCM tasks receive iRMX 88 messages from the iRMX 510 handler. The procedure for receiving iRMX 88 messages from the iRMX 510 handler is shown in Figure 9-11.

```
handler_message_ptr = RQ$WAIT           (response_mailbox,
                                        time_limit);
```

Figure 9-11. Procedure for an iRMX™ 88 Executive to Receive Messages for a DCM System

NOTE

Refer to the iRMX 88 REFERENCE MANUAL for more information about how to use the iRMX 88 call in Figure 9-11.

The definitions for the parameters in Figure 9-11 are as follows:

handler_message_ptr	A pointer to the iRMX 88 message as defined in Figure 9-9.
response_mailbox	The pointer to the response exchange you set in the original message.
time_limit	The amount of time the system waits for the response to arrive. A value of zero (0) tells the system to wait forever.

9.4.3 iPDS™/ISIS-II INTERFACE

The iRMX 510 handler for the iPDS Personal Development System allows users to take advantage of the services a DCM system offers. This handler passes iRMX 51 messages to and from the iSBX 344 parallel interface through procedural calls.

To transmit a message through the parallel interface, you must call CQDCMTRANSMIT with a parameter which points to a memory area containing the iRMX 51 message you wish to send. After the iRMX 510 handler completes the transmission, it returns control of the system to the program or procedure.

To receive a message through the parallel interface, you must call CQDCMRECEIVE with a parameter which points to a memory area containing the received iRMX 51 message. After the iRMX 510 handler completes the transmission, it returns control of the system to the program or procedure. Because iPDS calls are synchronous, one more call is necessary; CQDCMSTATUS$CHECK is a boolean function you can use to determine whether a message is waiting to be received from the iRMX 510 handler.

The following sections describe how to interact with the iRMX 510 handler.

9.4.3.1 Sending iPDS Messages to the iRMX™ 510 Handler

You can get information from an iPDS program to a DCM task by using iPDS calls to send messages to the iRMX 510 handler. The procedure for using iPDS calls to send information to the iRMX 510 handler is shown in Figure 9-11.

```
Call CQ$DCM$TRANSMIT          (iRMX_message_ptr);
```

Figure 9-12. Sending iPDS Messages to the iRMX 510 Handler

NOTE

Refer to the iPDS USER'S GUIDE for more information about how to use iPDS calls and procedures.

The definition for the parameter defined in Figure 9-12 is a follows:

iRMX_51_message_ptr	A pointer to the iRMX 51 message defined in Chapter 5.

9.4.3.2 Receiving iPDS Messages from the iRMX™ 510 Handler

DCM tasks can receive iPDS messages from the iRMX 510 handler. The procedure for receiving iPDS messages from the iRMX 510 handler is shown in Figure 9-13.

```
Call CQ$DCM$RECEIVE          (iRMX_51_buffer_ptr);
```

Figure 9-13. Receiving iPDS Messages from the iRMX™ 510 Handler

NOTE

Refer to the iPDS USER'S GUIDE for more information about how to use iPDS procedures.

The definition for the parameter in Figure 9-13 is as follows:

iRMX_51_buffer_ptr	A pointer to the buffer which receives iRMX 51 messages. This buffer is defined in Chapter 5 of this manual.

9.4.3.3 Checking for iRMX™ 51 Messages through the iRMX™ 510 Handler

The iPDS system checks for messages which are ready for the iRMX 510 handler to receive through the CQDCMSTATUS$CHECK call. The procedure for calling CQDCMSTATUS$CHECK is as shown in Figure 9-14.

```
status = CQ$DCM$STATUS$CHECK;
```

Figure 9-14. Procedure for Calling CQDCMSTATUS$CHECK

The definition for the parameter in Figure 9-14 is as follows:

status	A boolean byte that indicates whether a message is waiting for the iRMX 510 handler to accept it. TRUE means that a message is waiting to be accepted. FALSE means that no message is waiting.

9.5 CONFIGURING THE iRMX™ 510 INTERFACE

Configuring the iRMX 510 interface requires two steps. First, you must configure the interface hardware, then you must configure the supporting operating system. Note, that configuring the iRMX 86/286R and iRMX 88 Operating Systems is mandatory; configuring the iPDS ISIS-II Operating System is not required.

The following sections briefly describe how to configure the hardware interface and the supporting operating system. This manual assumes that you are familiar with configuring the supporting operating system. Refer to your operating system's configuration guide for more information.

9.5.1 CONFIGURING THE iRMX™ 510 INTERFACE TO WORK WITH AN iRMX™ 86/286R SYSTEM

You configure the iRMX 510 handler with respect to the underlying hardware which in this case is an iRMX 86/286R system. You must fill in the data areas of a hardware interface file in order to configure the hardware interface so that it works correctly with an iRMX 86 system.

You must create a file of configuration parameters, compile it, and link the resulting object file to the job in order to configure an iRMX 86 system so that it works correctly with the iRMX 510 handler.

9.5.1.1 Configuring the Hardware Interface

You can use the file, U51086.P86 to configure the hardware interface so that you can use the iRMX 510 handler with an iRMX 86 system. Figure 9-15 contains U51086.P86; you must fill in the described data areas, compile the file, and link it with the R86510.LIB library during the linking process described later in this chapter. Refer to Chapter 12 for the port addresses you need to supply in this file.

```
/*-----------------------------------------------------------------------
/*
/*    TITLE:          iRMX 510 - USER CONFIGURATION TEMPLATE
/*
/*    RELEASE:        1.0
/*
/*    DESCRIPTION:    iRMX 510 IS A SET OF DRIVERS FOR THE iSBX 344.
/*                    THIS FILE IS A TEMPLATE FOR THE USER TO CONFIGURE
/*                    AN iRMX 510 DRIVER INTO AN iRMX 86/286R SYSTEM.
/*                    THE USER FILLS IN THE DESCRIBED DATA ITEMS,
/*                    COMPILES THE FILE AND LINKS IT WITH THE R86510.LIB
/*                    LIBRARY DURING CONFIGURATION OF THE USER'S JOB.
/*-----------------------------------------------------------------------
```

```
U51086:
 DO;

/*I/O PORTS ON SBX */

  DECLARE Ufg_in_Fifo_Status_Port       BYTE       PUBLIC  DATA(OFOH);
  DECLARE Ufg_in_Fifo_Data_Port         BYTE       PUBLIC  DATA(OFOH);
  DECLARE Ufg_in_Fifo_Command_Port      BYTE       PUBLIC  DATA(OFOH);

  DECLARE Ufg_Out_Fifo_Status_Port      Byte       PUBLIC  DATA(OFOH);
  DECLARE Ufg_Out_Fifo_Data_Port        BYTE       PUBLIC  DATA(OFOH);
  DECLARE Ufg_Out_Fifo_Command_Port     BYTE       PUBLIC  DATA(OFOH);

/* INTERRUPT LEVELS */

  DECLARE Ufg_Recv_Interrupt_Level      BYTE       PUBLIC  DATA(048H);
  DECLARE Ufg_Xmit_Interrupt_Level      BYTE       PUBLIC  DATA(058H);

/*TASK PRIORITIES*/

  DECLARE Ufg_Recv_Task_Priority        BYTE       PUBLIC  DATA(065);
  DECLARE Ufg_Xmit_Task_Priority        BYTE       PUBLIC  DATA(081);

/*TASK STACK LENGTHS*/

  DECLARE Ufg_Recv_Task_Stkln           WORD       PUBLIC  DATA(200H);
  DECLARE Ufg_Xmit_Task_Stkln           WORD       PUBLIC  DATA(200H);

END:
```

Figure 9-15. Contents of U51086.P86

9.5.1.2 Configuring the iRMX™ 86/286R System

You must use the ICU to configure your iRMX 86/286R system so that it includes the configuration items listed in Figure 9-16; you need these items so that you can use an iRMX 86/286R system with the iRMX 510 handler. Refer to the iRMX™ 86 CONFIGURATION GUIDE for more information about configuring iRMX 86 systems.

```
Object Sys Calls
  (CO) Catalog Object                    Yes
  (LO) Lookup Object                     Yes

Job and Task Sys Calls
  (CT) Create Task                       Yes
  (DT) Delete Task                       Yes

Exchange Sys Calls
  (MBX) Create, Send and Receive Mailbox   Yes
  (DMB) Delete Mailbox                   Yes

Free Space Sys Calls
  (CS) Create Segment                    Yes
  (DS) Delete Segment                    Yes
  (GS) Get Size                          Yes

Interrupt Sys Calls
  (SI) Set Interrupt                     Yes
  (RI) Reset Interrupt                   Yes
  (EXI) Exit Interrupt                   Yes
  (SWI) Signal and Wait Interrupt        Yes
  (ENA) Enable Interrupt                 Yes
  (DSA) Disable Interrupt                Yes
  (GL) Get Level                         Yes

User Job
  (ODS) Object Directory Size [0-0FF0H]          0000H
  (PMI) Pool Minimum [20H - 0FFFFH]              0030H
  (PMA) Pool Maximum [20H - 0FFFFH]              FFFFH
  (MOB) Maximum Objects [1 - 0FFFFH]             FFFFH
  (MTK) Maximum Tasks [1 - 0FFFFH]               3H
  (MPR) Maximum Priority [0 - 0FFH]  *** Interrupt Usage Dependent***
  (AEH) Address of Exception Handler [CS:IP]     0000H:0000H
  (EM) Exception Mode [Never/Prog/Enviton/All]   Never
  (PV) Parameter Validation [Yes/No]             Yes
  (TP) Task Priority [0 - 0FFH]      *** Interrupt Usage Dependent ***
  (TSA) Task Start Address [CP:IP]   *** iRMX_510_Interface_Task ***
  (DSB) Data Segment Base [0 - 0FFFFH]           0000H
  (SSA) Stack Segment Address [SS:SP]            0000H:0000H
  (SS) Stack Size [0 - 0FFFFH]                   0200H
  (NPX) Numeric Processor Extension Used [Yes/No]   No
```

Figure 9-16. Configuration Items for iRMX™ 86/286R Systems

NOTE

The variable "iRMX_510_Interface_Task" is a public variable that is set when the iRMX 510 job is linked and located. You must substitute the value you obtained after the iRMX 510 job has been located.

9.5.1.3 Linking the iRMX™ 510 Interface Job to an iRMX™ 86/286R System

After you modify and compile U51086.P86, you must add U51086.OBJ and the iRMX 510 library (R51086.LIB) to the iRMX 86/286R job link list. By adding these files to your iRMX 86/286R job link list, you create the iRMX 510 interface job. You must also add the linked and located user jobs for the iRMX 510 handler and (optionally) applications to the list of files that LIB86 adds to the iRMX 86/286R library. You can do this by modifying the SUBMIT file generated by the ICU. Refer to the iRMX™ 86 CONFIGURATION GUIDE for information about locating jobs.

9.5.2 CONFIGURING THE iRMX™ 510 INTERFACE TO WORK WITH AN iRMX™ 88 EXECUTIVE

You configure the iRMX 510 handler with respect to the underlying hardware which in this case is an iRMX 88 system. You must fill in the data areas of a hardware interface file in order to configure the hardware interface so that it works correctly with an iRMX 88 system.

You must create a file of configuration parameters, compile it, and link the resulting object file to the job in order to configure an iRMX 88 system so that it works correctly with the iRMX 510 handler.

9.5.2.1 Configuring the Hardware Interface

You can use the file, U51088.P86 to configure the hardware interface so that you can use the iRMX 510 handler with an iRMX 88 system. Figure 9-17 contains U51086.P88; you must fill in the described data areas, compile the file, and link it with either the R5108L.LIB library or the R5108C.LIB during the linking process described later in this chapter. Refer to Chapter 12 for the port addresses you need to supply in this file.

```
/*-------------------------------------------------------------------
/*  TITLE:          iRMX 510 - USER CONFIGURATION TEMPLATE
/*
/*  RELEASE:        1.0
/*
/*  DESCRIPTION:    iRMX 510 IS A SET OF DRIVERS FOR THE iSBX 344.
/*                  THIS FILE IS A TEMPLATE FOR THE USER TO CONFIGURE
/*                  AN iRMX510 DRIVER INTO AN iRMX 88 SYSTEM.  THE
/*                  FILE AND LINKS IT WITH THE P.5108 L.LIB OR R5
/*                  LIBRARIES DURING CONFIGURATION OF THE USER'S JOB.
/*
/*-------------------------------------------------------------------

U51088:
    DO;

/*I/O PORTS ON SBX*/
  DECLARE Ucfg_in_Fifo_Status_Port      BYTE      PUBLIC  DATA(084H);
  DECLARE Ucfg_in_Fifo_Data_Port        BYTE      PUBLIC  DATA(080H);
  DECLARE Ucfg_in_Fifo_Command_Port     BYTE      PUBLIC  DATA(082);

  DECLARE Ucfg_Out_Fifo_Status_Port     BYTE      PUBLIC  DATA(084H);
  DECLARE Ucfg_Out_Fifo_Data_Port       BYTE      PUBLIC  DATA(080H);
  DECLARE Ucfg_Out_Fifo_Command_Port    BYTE      PUBLIC  DATA(082H);

/*INTERRUPT LEVELS*/

  DECLARE Ucfg_Recv_Interrupt_Level     BYTE      PUBLIC  DATA(04H);
  DECLARE Ucfg_Xmit_Interrupt_Level     BYTE      PUBLIC  DATA(05H);

/*INTERRUPT EXCHANGES USED: EXTERNAL REFERENCE AND ACCESS*/

  DECLARE RQL4EX BYTE EXTERNAL;
  DECLARE Ucfg_Xmit_Interrupt_Exchange  POINTER   PUBLIC  DATA(@RQL4EX);

  DECLARE RQL5EX BYTE EXTERNAL;
  DECLARE Ucfg_Xmit_Interrupt_Exchange  POINTER   PUBLIC  DATA(@RQL5EX);

END;
```

Figure 9-17. Contents of U51088.P86

9.5.2.2 Configuring the iRMX™ 88 Executive

You must use the ICU to configure your iRMX 88 system so that it includes the configuration items listed in Figure 9-18; you need these items so that you can use an iRMX 88 system with the iRMX 510 handler. Refer to the iRMX™ 88 REFERENCE MANUAL for more information about configuring iRMX 86 systems.

```
Task Name - DCMXMT
  Entry Point - DCM_Transmit_Task
  Stack Name - none
  Stack Size - 512
  Task Data Segment - dgroup
  Priority - 150
  Default Exchange - none
  Task Descriptor Name - none
  80877 NDP - No

Task Name - DCMRCV
  Entry Point - DCM_Receive_Task
  Stack Name - none
  Stack Size - 512
  Task Data Segment - dgroup
  Priority - 150
  Default Exchange - none
  Task Descriptor Name - none
  8087 NDP - No

Exchange Name - DCM_XMIT_EXCHANGE
  Scope - PUBLIC
  Interrupt Level - none

Exchange Name - DCM_RECV_EXCHANGE
  Scope - PUBLIC
  Interrupt Level - none
```

Figure 9-18. Configuration Items for iRMX™ 88 Systems

9.5.2.3 Linking the iRMX™ 510 Interface Job to an iRMX™ 88 System

After you modify and compile U51088.P86, you must add U51088.OBJ and either the iRMX 510 library, R5108C.LIB (for compact model) or the iRMX 510 library, R5108L.LIB, (for large model) to the iRMX 88 system link list. You must add these files to the link list through the interactive configuration utility for the iRMX 88 Executive (ICU88). Refer to the iRMX™ 88 REFERENCE MANUAL for information about using ICU88.

9.5.3 CONFIGURING THE iRMX™ 510 INTERFACE TO WORK WITH AN iPDS™ SYSTEM

You configure the iRMX 510 handler with respect to the underlying hardware, which in this case is an iPDS system. The iRMX 510 interface requires no hardware configuration. Refer to Chapter 12 of this manual for information about the iSBX 344 port for the iPDS system.

The iRMX 510 interface does not require that you modify the iPDS system. However, you can link the following files to create a user object module:

- R51085.LIB
- PLM.LIB
- SYSPDS.LIB

***

CHAPTER 10
INTRODUCTION TO THE DCM HARDWARE

10.1 OVERVIEW OF DCM HARDWARE

This chapter defines the relationship between the BITBUS interconnect and the hardware of the Distributed Control Modules. It also introduces the specific DCM board products.

The BITBUS interconnect provides a high-speed serial control bus for hierarchical systems. The devices that are connected to the BITBUS interconnect are called nodes. A BITBUS system consists of one master node and up to 250 slave nodes. The master node controls all the BITBUS interconnect operations while the slave nodes act as repliers. These nodes can have many forms, from simple process I/O points to highly intelligent I/O controllers. Refer to the BITBUS section of the DISTRIBUTED CONTROL MODULES DATA BOOK for more detailed discussions of the elements of a BITBUS system.

The DCM product line contains hardware that enables users to construct simple, programmable, high-speed, multi-microcontroller systems to use with the BITBUS interconnect. The DCM board products center around a common group of hardware and firmware elements called the DCM "core". The core, based on the DCM Controller, creates an intelligent, programmable, and very versatile BITBUS node. The following section describes the features of the DCM core. The subsequent sections describe the two specific DCM board products: the iSBX 344 BITBUS Controller MULTIMODULE board and the iRCB 44/10 Remote Controller Board.

Figure 10-1 illustrates the DCM hardware in a multidrop application.

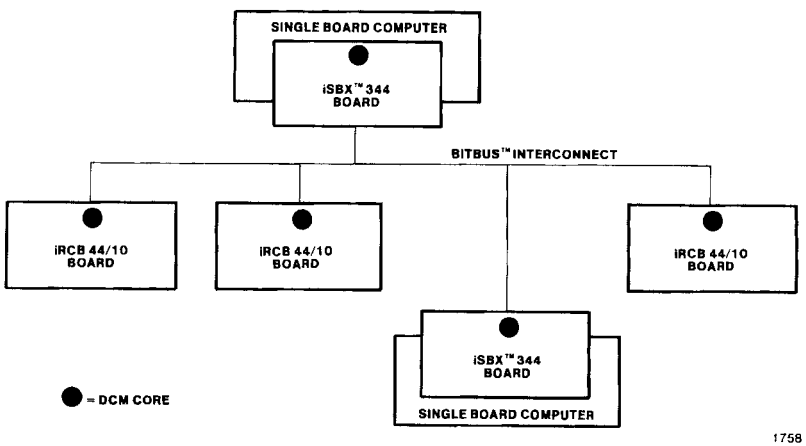


Figure 10-1. DCM Hardware Environment

10.2 THE DCM CORE

The DCM core consists of hardware and firmware elements that form the very powerful BITBUS node upon which both DCM board products are based. The core is not an Intel board product; it consists of the major components and the circuitry that is common to both the iSBX 344 board and the iRCB 44/10 board. Figure 10-2 illustrates the major elements of the DCM core. These elements include the DCM Controller, a transceiver interface, a decoder, an address latch, RAM, ROM, and a configuration section. Chapter 11 describes these elements in detail and provides information on the design of the core. In addition, Chapter 11 discusses the minimum requirements for designing DCM Controller-based BITBUS nodes.

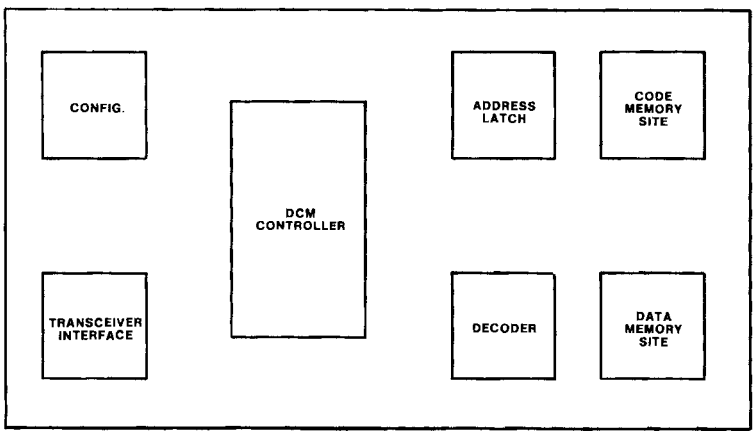


Figure 10-2. Elements of the DCM Core

The board products of the DCM product line feature a preprogrammed 8044 microcontroller. The 8044 with the on-chip firmware is called the DCM Controller. The 8044 is designed to serve as a programmable remote peripheral or peripheral subsystem controller. It contains its own CPU (the 8051 microcontroller), program memory, data memory, parallel I/O, timers/counters, and an intelligent serial interface that can serve as a primary or secondary station. The firmware, which the software chapters of this manual discuss, consists of the iRMX 51 Operating System, power-up tests, a communications gateway, and a Remote Access and Control Function.

10.3 THE iSBX™ 344 BITBUS™ CONTROLLER MULTIMODULE™ BOARD

The iSBX 344 BITBUS Controller MULTIMODULE board is a DCM core-based board that provides BITBUS interconnect access to all Intel products that support the iSBX I/O Expansion Bus. In addition to the core, the iSBX 344 board features a byte-deep FIFO, full duplex interface to the iSBX I/O Expansion bus and limited DMA operations. Figure 10-3 illustrates the major functional blocks of the iSBX 344 board.

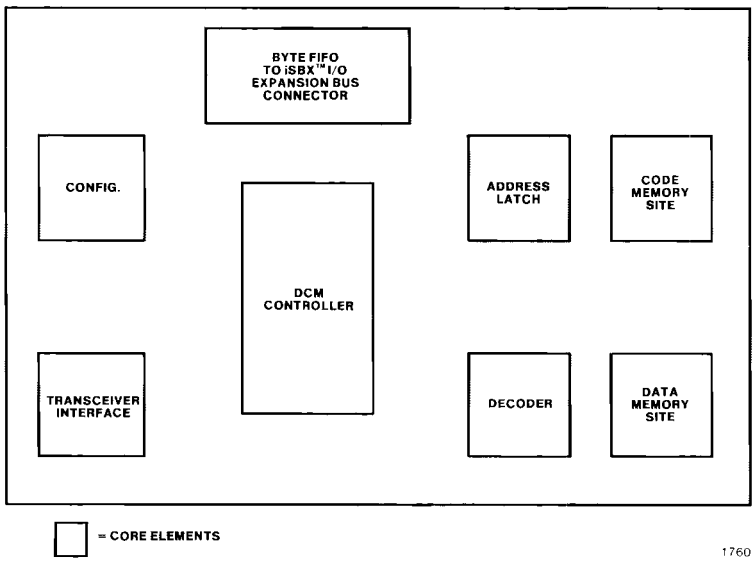


Figure 10-3. Elements of the iSBX™ 344 Board

10.4 THE iRCB 44/10 REMOTE CONTROLLER BOARD

The iRCB 44/10 Remote Controller Board is a single-high Eurocard form factor single board computer based on the DCM core. The iRCB 44/10 board also features iSBX I/O Expansion bus connector for I/O expansion and 24 parallel I/O lines. The iRCB 44/10 board also includes circuitry that enables users to add an on-board BITBUS repeater. Figure 10-4 illustrates the major functional blocks of the iRCB 44/10 board.

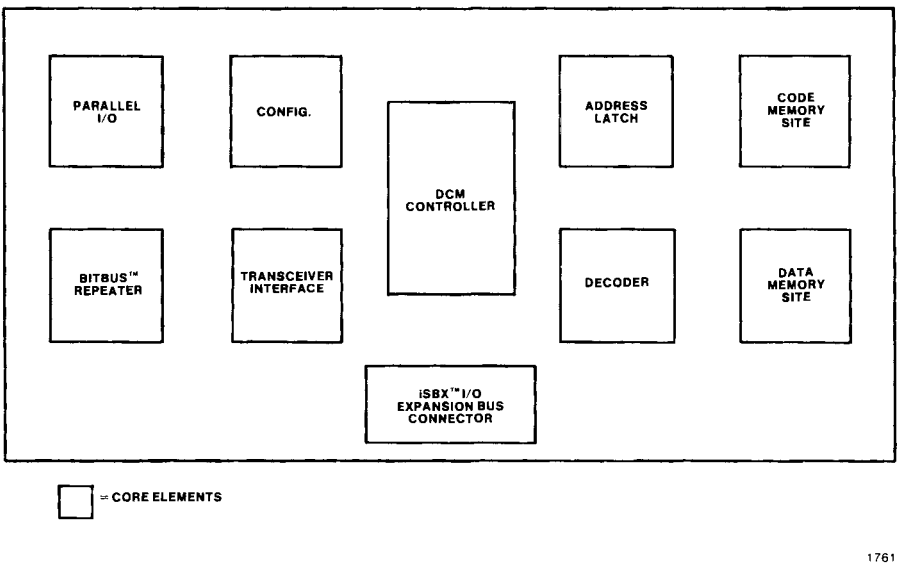


Figure 10-4. Elements of the iRCB 44/10 Board

***

CHAPTER 11
THE DCM CORE

11.1 INTRODUCTION

This chapter discusses the core section of the DCM hardware. The term core, as used in this manual, is the basic combination of hardware and firmware elements that are common to the boards in the DCM product line. The core forms a very versatile and powerful BITBUS node. It is important to realize that the core is the specific BITBUS node that Intel chose to use in the DCM board products. A designer need not be restricted to this configuration.

The main purpose of this chapter is to serve as a design-by-example design guide by describing Intel's DCM core in detail. However, because the core is only one of many possible ways to create a BITBUS node, this chapter also specifies the minimum requirements for any DCM Controller-based board to operate on the BITBUS interconnect. Initially, this chapter presents an overview of the basic elements that make up the core. Following the overview is a more detailed discussion of the design of the core and of all DCM Controller-based BITBUS nodes.

The discussions in this chapter assume that the reader is familiar with the BITBUS interconnect concept and the elements of the BITBUS interconnect. Refer to the BITBUS section of the DISTRIBUTED CONTROL MODULES DATA BOOK for details.

11.2 OVERVIEW OF CORE ELEMENTS

This section briefly describes the functional blocks of the DCM core. The DCM core consists of the following six functional areas:

- DCM Controller
- Address Latch
- Decoder
- External Memory Sites
- Serial Interface
- Configuration

Figure 11-1 illustrates the core.

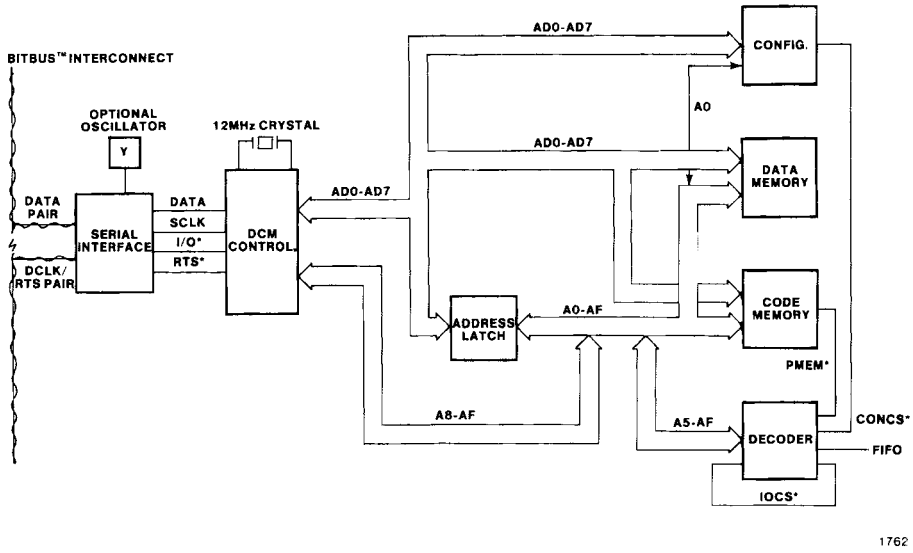


Figure 11-1. DCM Core Block Diagram

11.2.1 DCM CONTROLLER

The DCM Controller is the combination of the 8044 microcontroller and the DCM firmware. The DCM firmware resides in the internal code memory of the 8044 component. The firmware includes power-up tests, the iRMX 51 Executive, parallel and serial communications functions, and a Remote Access and Control (RAC) function. The RAC function is a preconfigured iRMX 51 task. The software section and subsequent parts of the hardware section of this manual provide more information about this firmware. The 8044 consists of an 8051 microcontroller CPU, a Serial Interface Unit (SIU), 192 bytes of on-chip RAM, and 4K bytes of on-chip ROM.

11.2.2 ADDRESS LATCH

The address latch demultiplexes the address/data bus for interfacing to standard memory and I/O devices.

11.2.3 DECODER

The decoder is a programmed PAL (Programmable Array Logic) that sets up the address spaces for the various devices in the system.

11.2.4 EXTERNAL MEMORY SITES

The core contains two 28-pin JEDEC universal memory sites that allow you to add external code and data memory. One site is dedicated to RAM for data storage. This site accepts 2k x 8 through 32k x 8 SRAMS only. The other memory site is dedicated to program storage. This site accepts various sizes of ROM, EPROM, SRAM, NVRAM, and EEPROM devices.

Section 11.3.4 contains all of the information you need to select and install memory components into these memory sites.

11.2.5 SERIAL INTERFACE

The serial interface consists of an RS485 driver and receiver and an optional clock source for synchronous operation. This interface supports both synchronous and self-clocked modes of operation.

11.2.6 CONFIGURATION

The configuration section of the core, which the DCM firmware requires, contains two registers used for configuration information: the mode register and the node address register. The mode register determines the mode of operation for the BITBUS interconnect; the node address register determines the node address for the DCM Controller.

11.3 DCM CORE DESIGN AND BITBUS™ NODE REQUIREMENTS

This section details the specific design of each element of the core. It also presents the specifications that a designer must consider when constructing the core. The discussions of each core element include the minimum requirements for building a DCM Controller-based BITBUS node.

11.3.1 HARDWARE REQUIREMENTS OF THE DCM CONTROLLER

The DCM firmware, which consists of the iRMX 51 Operating System, a communications gateway, power-up tests, and the Remote Access and Control function, assumes that certain interfaces and hardware surround the 8044. These external hardware interfaces are essential to the architecture and operation of the DCM firmware. This section discusses the hardware and the interfaces that the DCM Controller requires. This section does not apply to core designs that do not use the DCM Controller.

11.3.1.1 12 MHz Crystal

The DCM Controller requires a 12 MHz crystal for proper operation. The baud rate generation clock uses the 12MHz frequency to produce the 375kb/sec and 62.5kb/sec bit rates for the serial interface. The internal oscillator of the 8044 also uses this frequency for the CPU clock, the timer clock, and the Address Latch Enable (ALE) signal.

11.3.1.2 Data Memory

The iRMX 51 Executive and the communications functions require external data storage. Master and slave nodes require different amounts of data memory. A master requires 1k bytes of data memory; a slave requires 256 bytes of data memory. These memory requirements do not include the memory requirements of application tasks. The firmware assumes that the RAM exists in the external data segment, starting at location 0000H. Refer to Section 11.3.4 for more information on the memory of the core.

11.3.1.3 Program Memory

Incorporating user tasks into the core requires program storage space. User tasks are configured from the external code segment location. Refer to Section 11.3.4 for more information on the memory of the core. Refer to Chapter 5 for more information about iRMX 51 tasks.

11.3.1.4 External Hardware for RS485 Transceiver

The communications functions of the DCM firmware assume that external hardware for a half-duplex, RS485 serial line (the BITBUS interconnect) is present. The serial interface section of this chapter discusses this hardware and its corresponding specifications.

11.3.1.5 Diagnostic LEDs

The DCM Controller executes a diagnostic Power-up Test. The Power-up Test of the DCM Controller, which runs at power-up and chip reset, assumes that LEDs are driven through two 8044 ports. The diagnostic LEDs serve two functions: power-up diagnostics and user diagnostics.

The Power-up Test contains four individual diagnostic tests. As each test executes successfully, the LEDs indicate the stage of the Power-up Test. Table 11-1 lists the sequence of the Power-up Test and the corresponding states of the LEDs. The LEDs are connected to outputs of Port 1 of the 8044. The red LED is connected to pin P1.0, and the green LED is connected to pin P1.1.

The tests are run in the order that Table 11-1 shows. If all tests are successful, iRMX 51 initialization is entered. If any test fails, the power-up test enters a halt loop; and the LEDs indicate the last test the 8044 successfully executed. If all tests are successful, the green LED is on and the red LED is off. After the Power-up Test finishes executing, the LEDs are available to the user.

Table 11-1. Power-Up Test Sequence

Test Sequence	State of Port After Test Completion	
	Red LED	Green LED
Power-on	On	On
Prior To Start Of Tests	Off	Off
Test 1 - Instruction Set Test	On	On
Test 2 - ROM Checksum Test	On	Off
Test 3 - Internal RAM Test	Off	Off
Test 4 - External RAM Test	Off	On

11.3.2 ADDRESS LATCH REQUIREMENTS

Boards can meet the minimum requirements of the BITBUS interconnect scheme without using demultiplexed parts. Therefore, an address latch is not essential for all BITBUS nodes. The core, however, requires an address latch to demultiplex the multiplexed bus of the DCM Controller. The address latch enables the DCM Controller to interface to standard demultiplexed memory and I/O devices. Figure 11-2 shows the core's address latch, which consists of a simple 8-bit latch with its enable input attached to the ALE signal from the 8044.

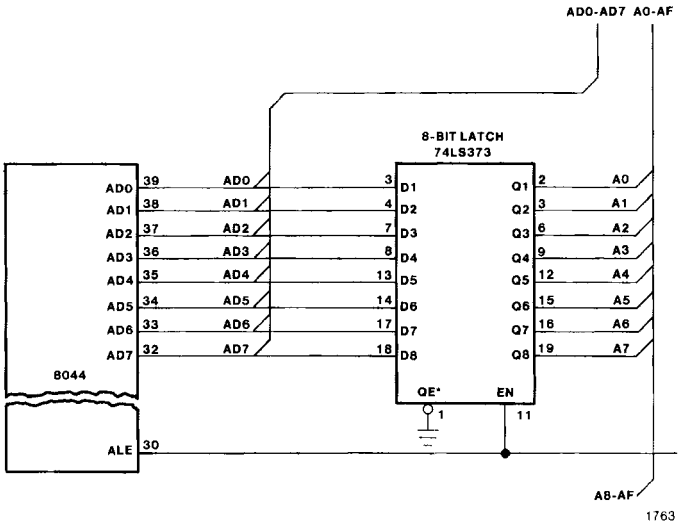


Figure 11-2. Core Address Latch

11.3.3 DECODER REQUIREMENTS

DCM Controller-based BITBUS nodes do not inherently require a decoder. The application and design of a node determines whether a decoder is necessary. The core, however, needs a decoder to set up the address spaces for the various devices in the core. A programmed PAL provides the core's decode function. Note that a PAL is only one means of satisfying the requirements for a decoder. The decoder, shown in Figure 11-3, uses 14 inputs to provide four outputs. Figure 11-3 contains the descriptions of the output signals. The data site decode signal is hardwired to the AF signal to enable the lower 32k of RAM only.

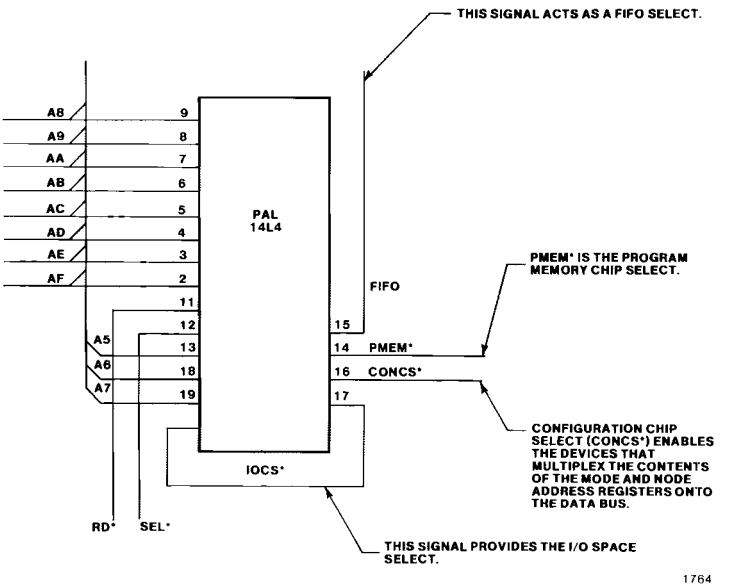


Figure 11-3. Decoder

Usually, you do not need to know the I/O port addresses since the firmware handles both the serial and parallel interfaces. For cases where you need to access to an I/O function directly, Table 11-2 lists the DCM Controller addressing restrictions for all implementations.

Table 11-2. DCM Controller I/O Addressing

Function	Address	Bit	Byte
Red LED (Port 0)	90H	X	
Green LED (Port 1)	91H	X	
TCMD	92H	X	
RFNE	B3H	X	
TFNF	B2H	X	
RDY/NE*	B4H	X	
Node Address	FFFFH		X
Configuration	FFFEH		X
Reserved	FFE0H-FFFDH		X
Digital I/O	FFC0H-FFDFH		X
SBX #4	FFB0H-FFBFH		X
SBX #3	FFA0H-FFAFH		X
SBX #2	FF90H-FF9FH		X
SBX #1	FF80H-FF8FH		X
User Defined	FF40H-FF7FH		X
Reserved	FF02H-FF3FH		X
FIFO Command	FF01H		X
FIFO Data	FF00H		X

11.3.4 MEMORY OF THE DCM CORE

The memory of the core consists of two major sections: internal and external. The internal memory resides in the DCM Controller; the external memory consists of the two 28-pin JEDEC universal memory sites. In addition, the 8044 component maintains separate address spaces for program memory and data memory. Therefore, the internal and external memory sections of the core each contain two subsections. These sections are as follows:

- Internal program memory
- Internal data memory
- External program memory
- External data memory

Figure 11-4 illustrates the memory map of the internal and external memory of the core.

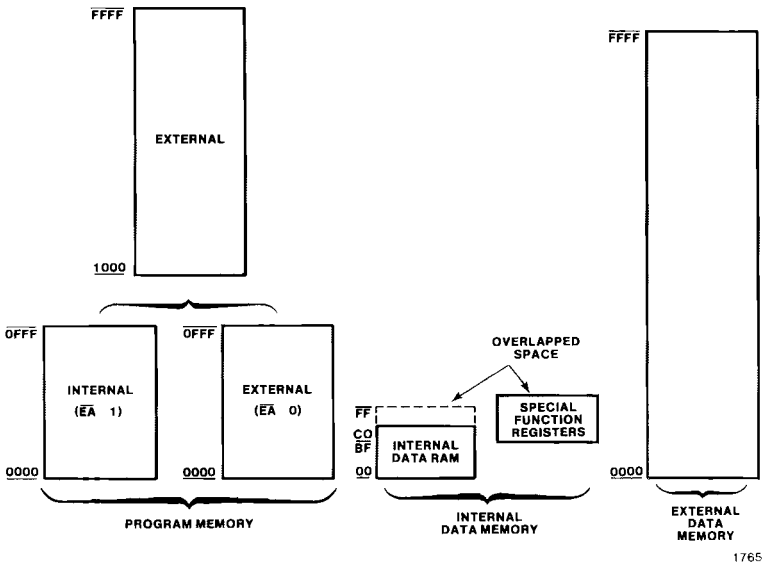


Figure 11-4. DCM Controller Memory Map

11.3.4.1 Internal Memory

The iRMX 51 Executive and DCM Controller requirements carefully ration the internal memory of the 8044. The internal program memory consists of 4k bytes of on-chip ROM. This on-chip ROM is used for code storage. The internal data memory consists of 192 bytes of on-chip RAM plus 35 special function registers. The iRMX 51 Executive allocates this on-chip RAM. Refer to the 8044 DATA SHEET for more information.

11.3.4.2 External Memory

The external memory of the core consists of two 28-pin sites: one dedicated to data storage and one dedicated to program storage. The 8044 is capable of addressing up to 64k bytes of external data memory. However, the memory site dedicated to data storage accepts 2k x 8 through 32k x 8 SRAMs only.

The 8044 is also capable of addressing up to 64k bytes of external program memory. The site dedicated to program storage accepts several types and sizes of memory devices. Although this site is dedicated to program storage, it is writable in the data memory space for program download. The next section provides more details about this feature.

11.3.4.3 Overview Of Memory Addressing Options

The DCM core offers two options for memory addressing. Both options maintain the 8044 architecture of separate code and data spaces.

The first option, option A, allows up to a 64kx8 device to occupy the code memory site for program storage. The DCM Controller requires that the Initial Task Descriptor (ITD) to start at FFF0H. Refer to Chapter 7 for more information on the ITD. Option A also supports the External Access (EA) option of disabling internal program store where code must start at 0000H. In addition, up to a 32kx8 SRAM device may be used for data memory store. In both cases, if devices are used which are smaller than the maximum allowable size, they are addressed multiple times. For example, if option A is selected, the address space for data memory is 32k bytes. If you insert a 2k x 8 device in the data site, the same 2k bytes of information is replicated in the other 15 2k-byte segments of the 32k byte data space.

Note that in both memory addressing options, the I/O addresses are mapped into the data memory space.

Figure 11-5 shows memory addressing option A.

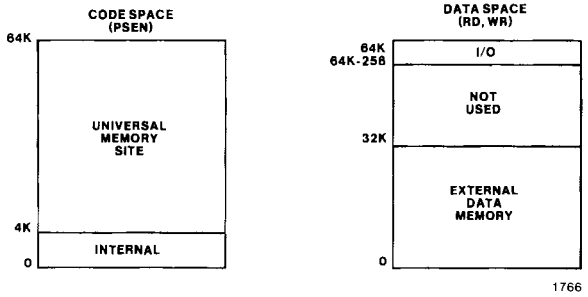


Figure 11-5. Memory Addressing Option A

Option B allows the same data memory configuration as option A while providing a more flexible program store. In this configuration, you can connect the write signal to the code memory site to allow program download. Note that in this mode, data is written into the data space but read from the code space for the code site. In other words, part of the data space is mapped to the code site. This allows you to write to the code site via the data space. The multiple addressing of devices that are less than the maximum allowable size also applies to option B.

Figure 11-6 shows memory addressing option B.

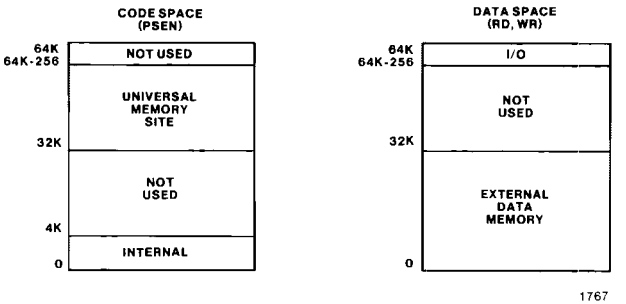


Figure 11-6. Memory Addressing Option B

11.3.4.4 Memory Component Selection And Installation

This section provides instructions for selecting and installing the user-provided memory devices. This section also provides the AC and DC specifications and timing information for the JEDEC memory sockets. You can install one of several types of memory components, including EPROM and static RAM devices, into the memory sockets of the core. Table 11-3 lists the types of byte-wide memory components that the core supports.

Table 11-3. Supported Byte-Wide Memory Devices

Memory Type	Memory Capacity	Data Site	Code Site
EPROM/ROM	4k x 8 - 64k x 8	No	Yes
SRAM	2k x 8 - 32k x 8	Yes	Yes
NVRAM	2k x 8 - 16k x 8	No	Yes
E2PROM	2k x 8 - 16k x 8	No	Yes

The type of memory component used must be compatible with the AC and DC specifications of the chip socket. Both JEDEC sites support devices with 200ns access time. All specifications, except option B in the code site, typically meet the 250ns access time specification. Option B in the code site always requires 200ns devices. Table 11-4 lists the specifications for the byte-wide sockets.

Table 11-4. Byte-Wide Socket AC And DC Specifications

Parameter	Conditions	Minimum	Maximum	Units
DC Characteristics				
V_{IL}			0.8	V
V_{IH}		2.0		V
I_{IL}	V_{IN}=0.4V		$\pm$10	uA
I_{IH}	V_{IN}=2.4V		$\pm$10	uA
V_{OL}	I_{OL}=2.1mA		0.45	V
V_{OH}	I_{OH}=-400uA	2.4		V
AC Characteristics				
C_{IO}	I/O Capacitance		12	pf

The memory devices used must also comply with the timing specifications of the JEDEC sockets. Table 11-5 provides the memory read operation timing specifications for both data site and the code site. Figure 11-7 provides corresponding timing diagram for the read operation.

Table 11-5. Memory Socket Read Timing

Parameters	Code Site		Data Site	
	Min.	Max.	Min.	Max.
t_1 - Add. stable to data valid		284ns		567ns
t_2 - CS/ stable to data valid		302ns (1)		585ns
t_3 - OE/ stable to data valid		125ns		250ns
t_4 - OE/ width (RD/)	215ns		400ns	
t_5 - Data hold after OE/ (RD/)	0		0	
t_6 - Add. stable after OE/ high	75ns		43ns	
t_7 - CS/ stable after OE/ high	75ns		43ns	97ns
t_8 - Data float after OE/ (RD/)		75ns		
Note: 1. In memory addressing option B, this value is 232ns.				

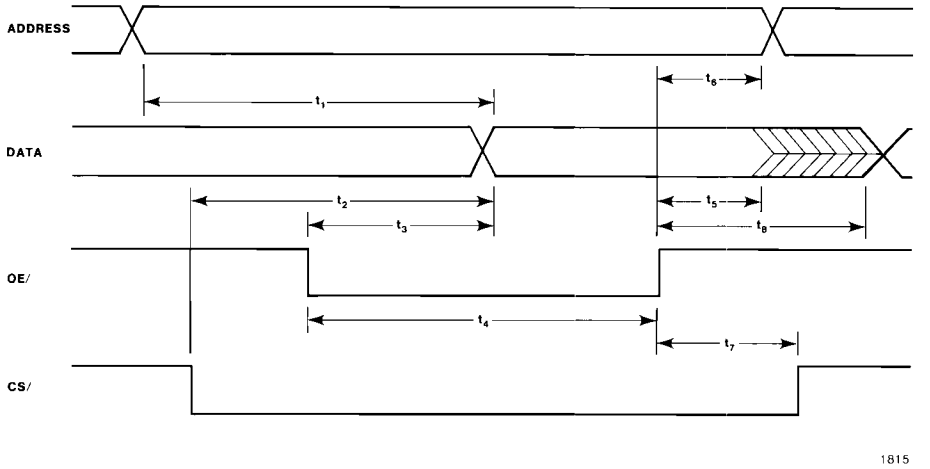


Figure 11-7. Universal Memory Site Read Timing

Table 11-6 provides the memory write operation timing specifications for the JEDEC sockets. Figure 11-8 shows the corresponding timing diagram for memory write operations.

Table 11-6. Memory Socket Write Timing (Both Sites)

Parameters	Minimum	Maximum
t_1 - Address stable to WR/ low	185ns	-
t_2 - Data stable to WR/ low	33ns	-
t_3 - CS/ stable to WR/ low	203ns	-
t_4 - Address stable after WR/ high	49ns	-
t_5 - Data stable after WR/ high	33ns	-
t_6 - CS/ stable after WR/ high	49ns	-
t_7 - WR/ width	400ns	-

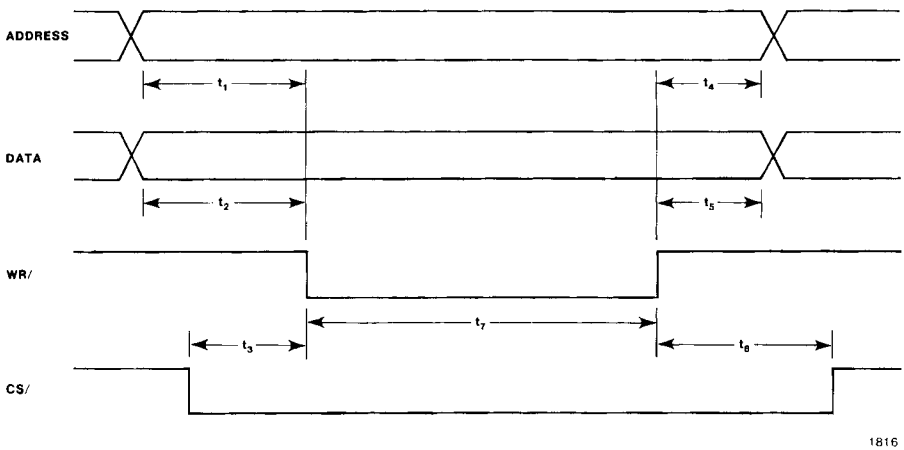


Figure 11-8. Universal Memory Site Write Timing

Figure 11-9 shows how to install either 24-pin or 28-pin memory chips into the JEDEC-compatible sockets.

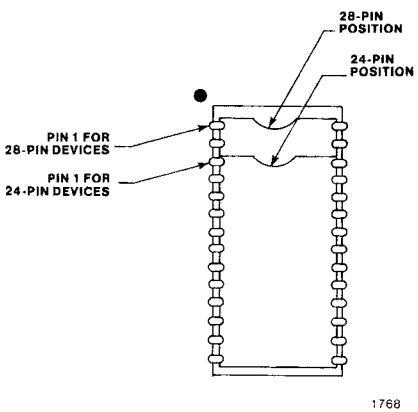


Figure 11-9. Memory Chip Installation

CAUTION

Never insert MOS components into a board when power is on. Doing so can damage the components.

CAUTION

All MOS components such as ROM, EPROM, and RAM are highly susceptible to damage from static electricity. Use extreme caution when installing MOS components in a low humidity environment. Always ground yourself before handling MOS components. This precaution ensures that a static charge build-up is not dissipated through or around the MOS devices.

11.3.4.3 Universal Memory Site Jumper Matrix Configuration

The core contains two jumper matrices for configuring the memory sockets: one for the code site and one for the data site. You can configure each matrix independently for different types of memory devices. The standard universal memory site jumper matrix consists of 15 stake pins arranged in two rows, as Figure 11-10 illustrates. The missing pin in the matrix serves as a key to the orientation of the matrix.

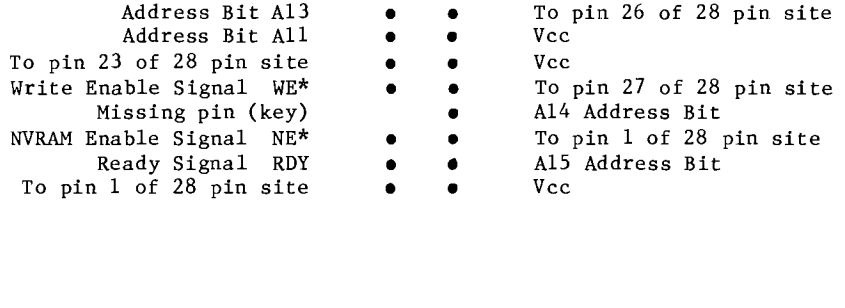


Figure 11-10. Memory Socket Jumper Matrix Configuration

You configure each matrix by installing jumpers required to place signals on the proper pins of the memory devices. Figures 11-11 through 11-14 provide the diagrams for the different configurations. The memory types which each figure illustrates are as follows:

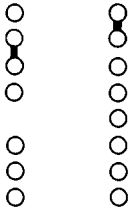
Figure	Device Type
Figure 11-11	EPROM/ROM devices
Figure 11-12	SRAM devices
Figure 11-13	EEROM devices
Figure 11-14	NVRAM devices

The code site accepts all of the devices outlined in these figures. The data site only accepts the SRAM devices.

4K x 8 EPROM
(example 2732A)

Pin1	NC
Pin 27	NC
Pin 26	Vcc
Pin 23	A11

JUMPER MATRIX CONFIGURATION



8K x 8 EPROM
(example 2764)

Pin 1	Vcc/Vpp
Pin 27	Vcc/PGM
Pin 26	NC
Pin 23	A11

JUMPER MATRIX CONFIGURATION

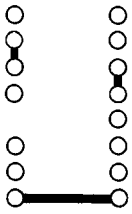
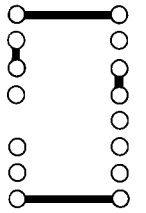


Figure 11-11. Universal Site Configuration For EPROM Devices

16K x 8 EPROM
(example 27128)

Pin 1	Vcc/Vpp
Pin 27	Vcc/PGM
Pin 26	A13
Pin 23	A11

JUMPER MATRIX CONFIGURATION

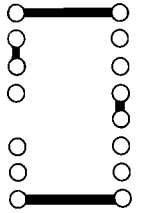


1769

32K x 8 EPROM
(example 27256)

Pin 1	Vcc/Vpp
Pin 27	A14
Pin 26	A13
Pin 23	A11

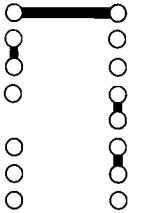
JUMPER MATRIX CONFIGURATION



64K x 8 EPROM
(example 27512)

Pin 1	A15
Pin 27	A14
Pin 26	A13
Pin 23	A11

JUMPER MATRIX CONFIGURATION



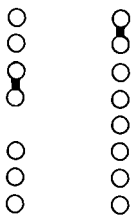
1770

Figure 11-11. Universal Site Configuration For EPROM Devices (continued)

2K x 8 STATIC RAM

Pin1	NC
Pin 27	NC
Pin 26	Vcc
Pin 23	WE*

JUMPER MATRIX CONFIGURATION

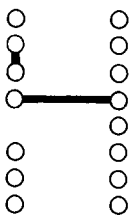


1771

4K x 8 STATIC RAM

Pin 1	NC
Pin 27	WE*
Pin 26	NC
Pin 23	A11

JUMPER MATRIX CONFIGURATION



8K x 8 STATIC RAM

Pin 1	NC
Pin 27	WE*
Pin 26	NC
Pin 23	A11

JUMPER MATRIX CONFIGURATION

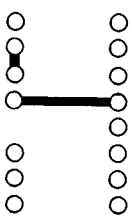
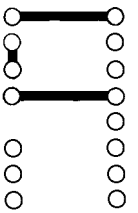


Figure 11-12. Universal Site Configuration For Static RAM Devices

16K x 8 STATIC RAM

Pin 1	NC
Pin 27	WE*
Pin 26	A13
Pin 23	A11

JUMPER MATRIX CONFIGURATION

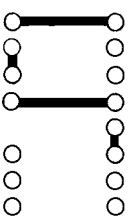


1772

32K x 8 STATIC RAM

Pin 1	A14
Pin 27	WE*
Pin 26	A13
Pin 23	A11

JUMPER MATRIX CONFIGURATION



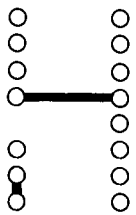
1773

Figure 11-12. Universal Site Configuration For Static RAM Devices (continued)

½K x 8 EEPROM AND 2K x 8 EEPROM

Pin 1	RDY
Pin 27	WE*
Pin 26	NC
Pin 23	NC

JUMPER MATRIX CONFIGURATION

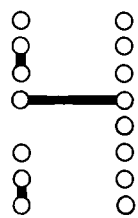


1774

4K x 8 EEPROM AND 8K x 8 EEPROM

Pin 1	RDY
Pin 27	WE*
Pin 26	NC
Pin 23	A11

JUMPER MATRIX CONFIGURATION



1775

16K x 8 EEPROM

Pin 1	RDY
Pin 27	WE*
Pin 26	A13
Pin 23	A11

JUMPER MATRIX CONFIGURATION

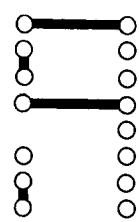
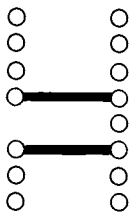


Figure 11-13. Universal Site Configuration For EEPROM Devices

½K x 8 NVRAM
1K x 8 NVRAM
AND
2K x 8 NVRAM

Pin 1	NE
Pin 27	WE*
Pin 26	NC
Pin 23	NC

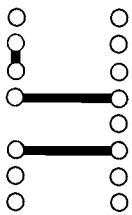
JUMPER MATRIX CONFIGURATION



4K x 8 NVRAM
AND
8K x 8 NVRAM

Pin 1	NE*
Pin 27	WE*
Pin 26	NC
Pin 23	A11

JUMPER MATRIX CONFIGURATION

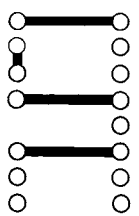


1776

16K x 8 NVRAM

Pin 1	NE*
Pin 27	WE*
Pin 26	A13
Pin 23	A11

JUMPER MATRIX CONFIGURATION



1777

Figure 11-14. Universal Site Configuration For NVRAM Devices

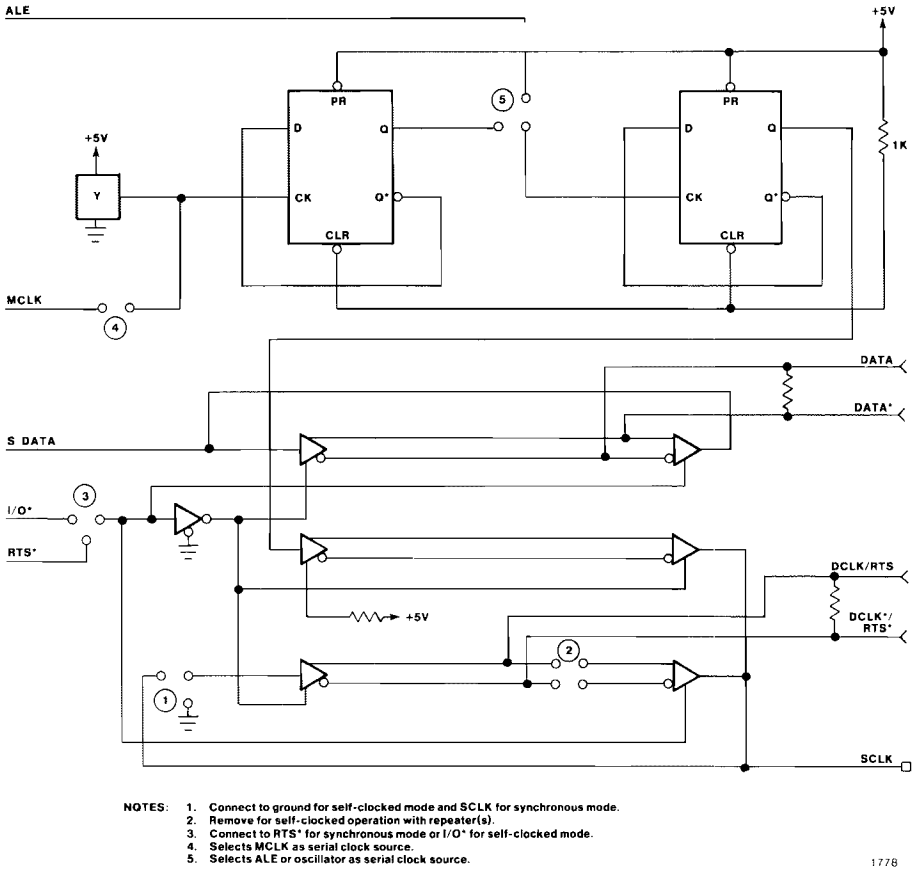
11.3.5 SERIAL INTERFACE REQUIREMENTS

The BITBUS interconnect requires that all BITBUS node transmitters meet the requirements of a RS485 generator. For more information on the minimum specifications of a BITBUS serial interface, refer to the BITBUS section of the DISTRIBUTED CONTROL MODULES DATA BOOK.

The serial interface for the core consists of a half-duplex RS485 transmitter/receiver and an optional clock source for synchronous operation. This interface supports both synchronous and self-clocked modes of BITBUS operation. The multiple-mode capability provides a wide range of performance and distance options for a variety of applications. In all cases, the BITBUS interconnect is two differential pairs of wires. The 8044 Data Sheet contains more details of the 8044 interface and Serial Interface Unit (SIU). Figure 11-15 shows the core's transceiver interface. This section describes the requirements of the interface for the two modes of operation.

11.3.5.1 Synchronous Operation

The DCM core supports two sources for the clock in the synchronous mode: Address Latch Enable (ALE) or oscillator. The ALE source costs less. This method uses the ALE output of the 8044 to generate a clock frequency of slightly less than 1 MHz. This is possible since the 8044 generates an ALE every six oscillator periods (2 MHz), except when an external data memory access is made. When the 8044 accesses external data memory access, it generates an ALE every twelve oscillator periods. As Figure 11-15 shows, this signal is run through a flip-flop, divide by two, to provide a signal which satisfies the 8044 required clock high and low times. The output is slightly less than 1 MHz due to the "missing ALE" on external data memory accesses; however, the only effect is a slight degradation in performance.



NOTES:
1. Connect to ground for self-clocked mode and SCLK for synchronous mode.
2. Remove for self-clocked operation with repeater(s).
3. Connect to RTS* for synchronous mode or I/O* for self-clocked mode.
4. Selects MCLK as serial clock source.
5. Selects ALE or oscillator as serial clock source.

Figure 11-15. Serial Interface

The oscillator option allows you to install a 2.0 MHz to 9.6 MHz crystal oscillator to generate a .5 MHz to 2.4 MHz serial clock. Although more expensive than the ALE option, this method can provide the highest performance.

It is important to understand that these options are on a node-by-node basis. The system does not require the use of a single option or a single frequency throughout. Also, the cabling of the data and clock signals influences the operation of a system. You should ensure that these lines have matched delays. Using the same type of cable and the same cable lengths aid in matching the delays.

11.3.5.2 Self-Clocked Operation

The self-clocked mode of operation allows long distance operation of the BITBUS interconnect. The self-clocked mode uses two differential signal pairs: one for data (DATA, DATA*) and one for transceiver control (RTS, RTS*). The data signal lines carry NRZI (non-return to zero inverted) encoded data. This encoding method combines clock and data onto the same signal pair. The transceiver control signal pair is used for transceiver control within repeaters. When repeaters are not present, the transceiver control pair can be omitted. With the addition of repeaters, the number of nodes and the distances can be further increased.

11.3.6 CONFIGURATION

DCM Controller-based BITBUS nodes require some form of configuration section that provides node address and mode parameters at a fixed external I/O location. The configuration section of the core contains two registers which provide the node address and the BITBUS mode for the DCM Controller. Intel allows you to change these functions via jumpers. This provides more flexibility in the core. Figure 11-16 shows the configuration section of the DCM core. Figure 11-17 illustrates the node address register, and Figure 11-18 illustrates the mode register.

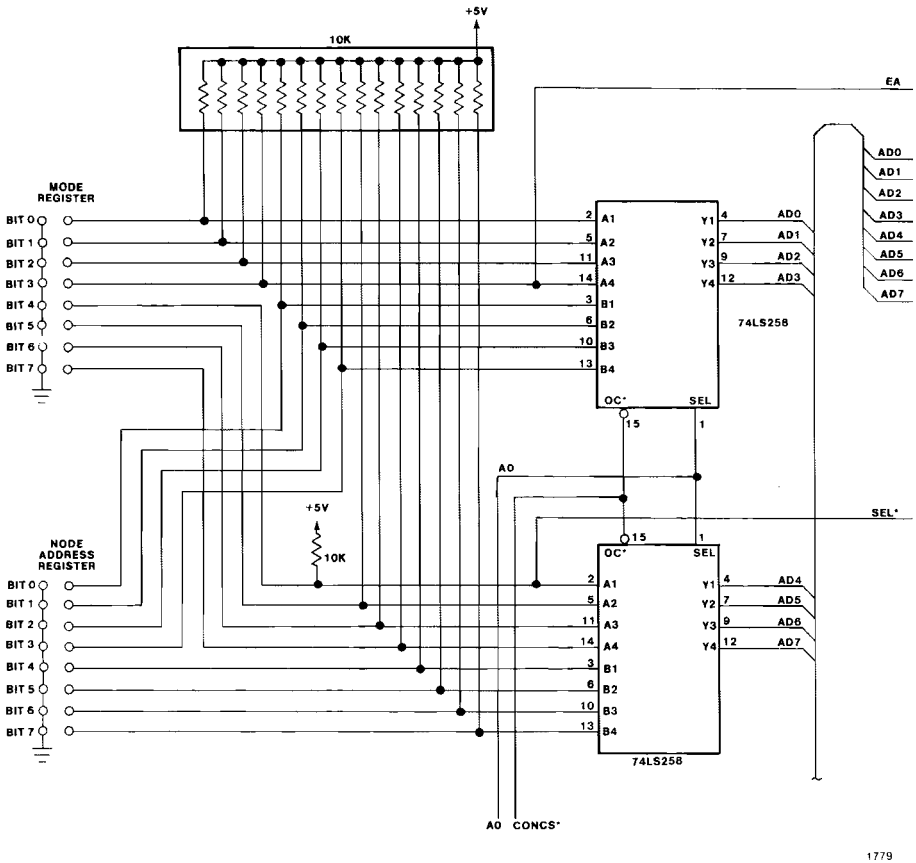


Figure 11-16. Configuration Section

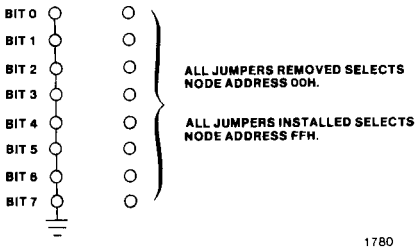


Figure 11-17. Node Address Register

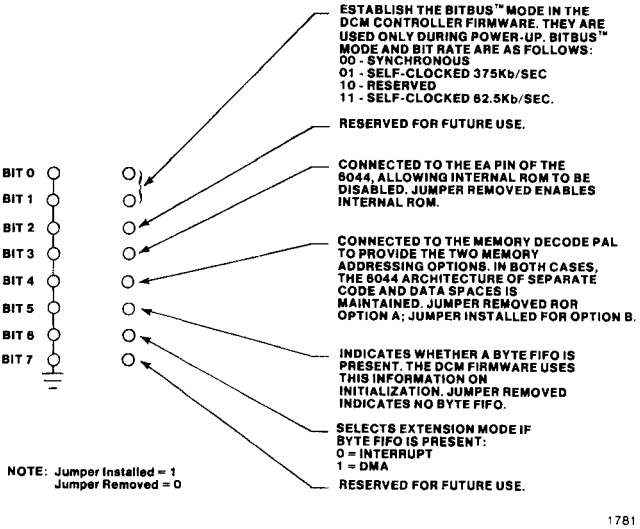


Figure 11-18. Mode Register

***

CHAPTER 12
USING THE iSBX™ 344 BITBUS™ CONTROLLER MULTIMODULE™ BOARD

12.1 INTRODUCTION

The iSBX 344 BITBUS Controller MULTIMODULE board is an 8044-based iSBX board designed to provide a BITBUS connection to all Intel products that support the iSBX I/O Expansion Bus. Figure 12-1 shows the iSBX 344 board, which is part of the Distributed Control Modules product line.

This chapter lists the key features of the iSBX 344 board and describes the configuration of each independent function on the board. The configuration information includes jumper information, addressing information, interface information, and programming information.

Because the iSBX 344 board is based on the DCM core, many of the discussions in this chapter refer you to Chapter 11 which provides a more in-depth operational discussion of the core. In order to avoid redundancy, this chapter is limited to configuration information and operational discussions of only the sections of the iSBX 344 board that are not part of the core.

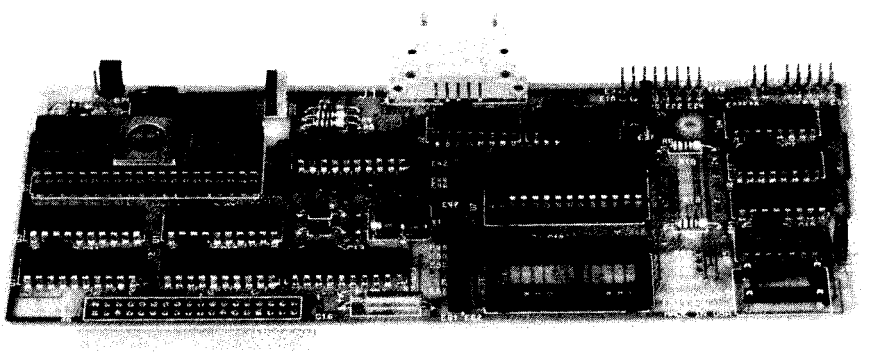


Figure 12-1. iSBX™ 344 BITBUS™ Controller MULTIMODULE™ Board

12.2 DESCRIPTION

This section provides a brief description of the key features of the iSBX 344 BITBUS Controller MULTIMODULE board. The iSBX 344 board, shown in Figure 12-2, consists of the DCM core enhanced with a Byte FIFO. This section reviews the DCM core and describes the Byte FIFO interface in detail.

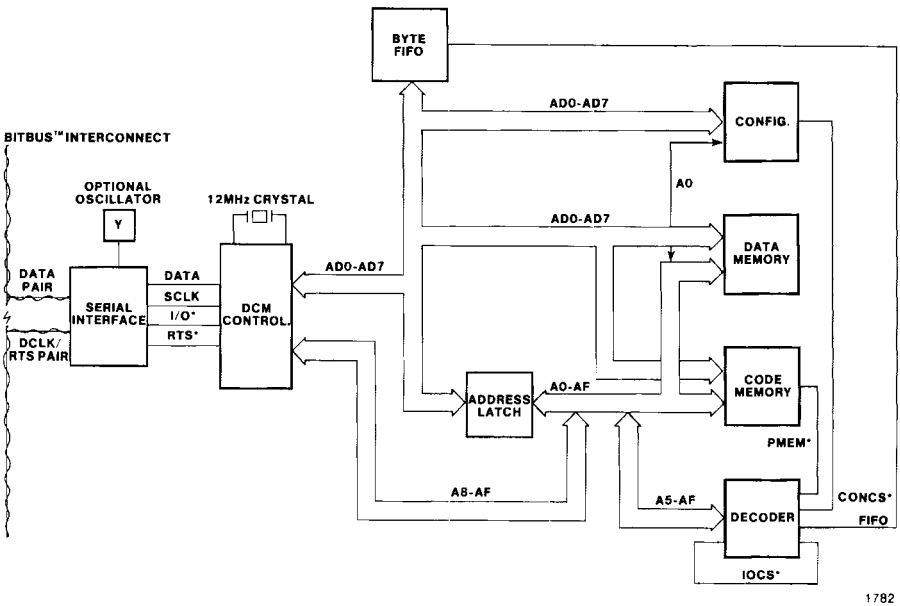


Figure 12-2. iSBX™ 344 Board Block Diagram

12.2.1 THE DCM CORE

The DCM core is the combination of hardware and software elements that are common to all boards in the DCM product line. The DCM core provides the foundation for the iSBX 344 BITBUS Controller MULTIMODULE board. As the areas outlined in red illustrate in Figure 12-2, the core consists of the DCM Controller, an address latch, a decoder, data memory space, code memory space, a serial interface, and a configuration section.

12.2.2 BYTE FIFO INTERFACE

DCM Controller-based BITBUS nodes do not require a parallel interface. However, the addition of a parallel interface allows expansion to other processors into the system.

The DCM system includes a byte-wide, First-In/First-Out (FIFO) parallel interface for communicating with either a master or a slave extension device. An extension device is a device attached to a BITBUS node via the byte FIFO. This feature allows a processor (or processor board) with different capabilities than the 8044 to be used as a part of a BITBUS node. The iSBX 344 board is specifically designed for this function.

The byte FIFO interface on the iSBX 344 board provides the required buffering between the 8044 and its extensions. Figure 12-3 shows this parallel interface. The interface supports both byte and message transfer protocol in hardware via three I/O ports (one for data, one for command, and one for status). The extension side of the interface (the iSBX connection on the extension device) supports polled, interrupt, and limited Direct Memory Access (DMA) modes of operation. The 8044 side supports only polled and interrupt modes of operation.

The byte FIFO interface requires hardware for two unidirectional, byte-wide FIFO queues which I/O read and write instructions can access. One queue is for byte transfer from the extension to the DCM Controller device, and the other is for byte transfer from the DCM device to the extension. The extension and the DCM Controller device access the queues through different 8-bit, parallel busses. Queues can have a depth of only one byte. However, the DCM firmware can run with 'n' byte queues. A byte may be enqueued or dequeued as either a command byte or data byte, depending upon the address written or read. Each queue maintains the sequentiality between command and data bytes. This creates a forced synchronization between command and data bytes, eliminating the possibility of misreading data as a command or a command as data.

A Status Byte, accessed through a third location, contains the operational status of the two queues. This status indicates whether the outgoing queue is full or not full. The status also indicates whether the incoming queue is empty or not empty; and if it is not empty, whether the next available byte is a command or data byte.

The extension device can exchange bytes via polls, interrupts, or limited DMA. Chapter 9 contains information on extension interfacing through the iSBX I/O Expansion bus to the iSBX 344 board.

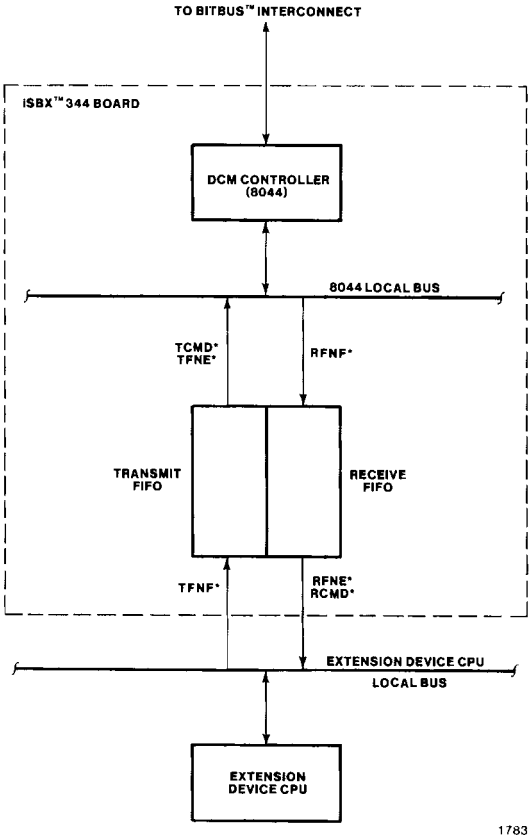


Figure 12-3. Byte FIFO Interface

The following sections discuss a few of the possible modes of operation for the byte FIFO interface. These modes are intended only as examples since there are many other possible protocols. For simplicity, only the extension side of the interface is discussed in detail. However, this discussion covers the 8044 side of the interface indirectly since the 8044 side supports a subset of the extension side functions. In all cases, the text explicitly states whether the 8044 provides support.

When using the DCM software, you do not need to understand the operation of the byte FIFO. If the DCM firmware is used, the 8044 side of the interface is not visible to the user. Similarly, if you use the DCM Support Package on the extension device, the byte FIFO interface is also transparent.

The following discussions use standard conventions for designating signal types. The absence (active high) or presence (active low) of an asterisk (*) indicates the active state of a signal. Also, all names begin with a "T" (transmit) or an "R" (receive) to indicate which FIFO is involved. The FIFO names and all the discussions in this section are always referenced with respect to the extension device accessing the 8044, as Figure 12-3 shows.

12.2.2.1 Polled Operation

The iSBX 344 board provides a read only status register to the extension device for polled operation. Both sides of the interface provide support for polled mode operation. This status register provides three signals: Transmit FIFO Not Full (TFNF*), Receive FIFO Not Empty (RFNE*), and Receive Command (RCMD*). This section describes these signals and details the sequence of typical polled mode send and receive operations. The signal names in the parentheses are the analogous signals on the 8044 side of the interface.

TFNF* (RFNF*) — The TFNF* bit becomes active whenever the transmit FIFO can accept a data or command byte from the extension device. When the transmit FIFO is full, this bit goes inactive (or high).

RFNE* (TFNE*) — The RFNE* bit becomes active whenever the receive FIFO has a valid data byte or command. Reading the proper port deactivates this bit until the next data byte or command becomes valid.

RCMD* (TCMD*) — The RCMD* signal, in addition to the RFNE* signal, goes active when the receive FIFO has a valid command. A read operation to the FIFO command port deactivates this bit. When RCMD* is active, reading the data port does not deactivate RCMD* or RFNE*. Likewise, reading the command port when RCMD* is inactive does not deactivate RFNE*. This feature is provided to guarantee proper message synchronization. This bit is only valid when RFNE* is active.

Figure 12-4 shows the sequence of a typical polled mode send. This operation is initiated when the extension device has a message to send. Before sending the first byte, the extension device reads the status register and checks the TFNF* bit. If inactive, the extension device continues to poll or returns to the calling routine indicating that the message was not sent. In any case, when an active TFNF* bit is detected, the extension device can send a byte. After sending all message bytes to the data port, the extension sends one final byte to the command port to indicate end of message. This final command byte must be 0, otherwise it resets the DCM Controller.

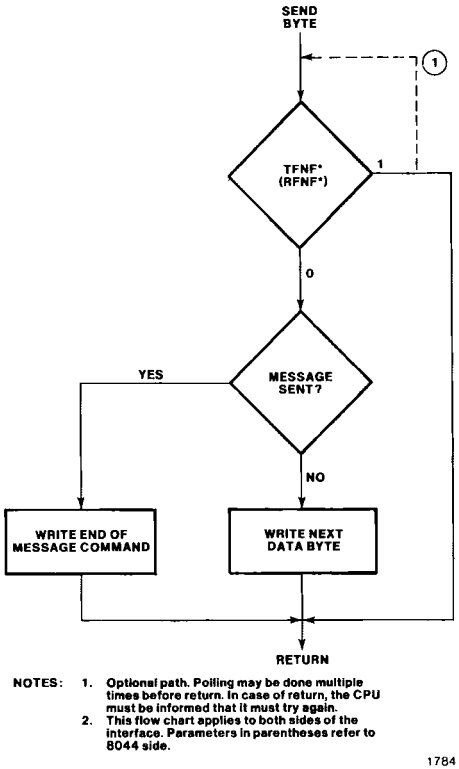
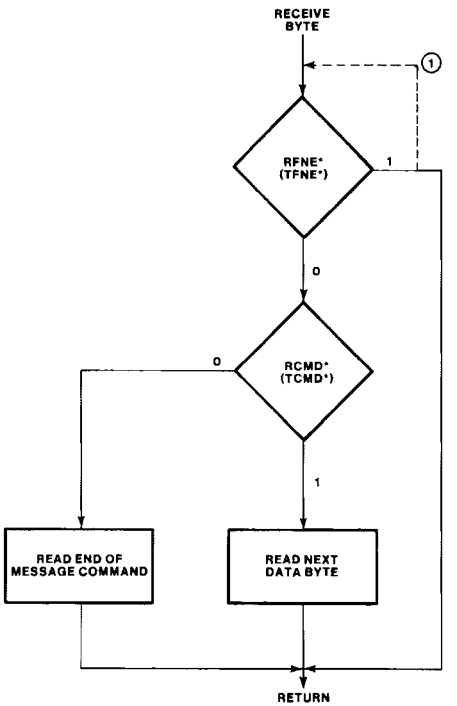


Figure 12-4. Polled Mode Send

Figure 12-5 shows the sequence of a typical polled mode receive. The extension device initiates the operation by polling the status register and finding the RFNE* bit active. The extension then tests the RCMD* bit for an end of message command. If the RCMD* bit is inactive, the extension device reads from the data port. If the RCMD* bit is active, the extension device reads from the command port.



NOTES:
1. Optional path. Polling may be done multiple times before return.
2. This flow chart applies to both sides of the interface. Parameters in parentheses refer to 8044 side.

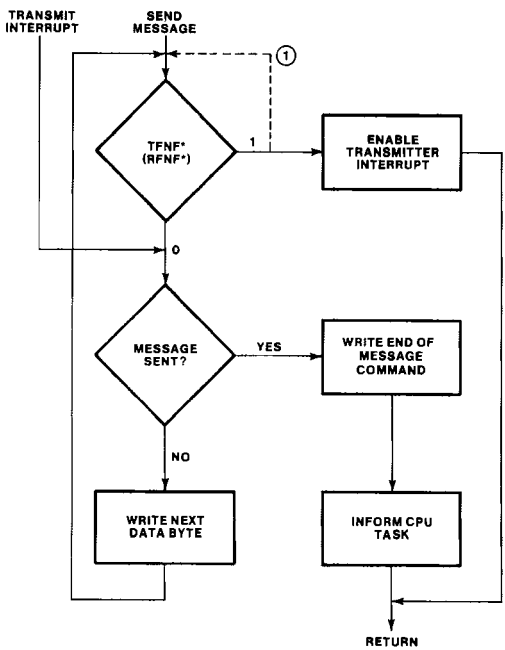
1785

Figure 12-5. Polled Mode Receive

12.2.2.3 Polled/Interrupt Mode

Figures 12-6 and 12-7 show the respective flow charts for send and receive operations in the Polled/Interrupt mode. This mode combines the advantages but avoids the weaknesses of the polled and interrupt modes. It provides a mode which is the most efficient for most applications. In polled mode the most significant time consumer is polling for messages that might not occur immediately. Interrupts provide one solution to this problem. Interrupts prevent unnecessary polling when activity is infrequent. However, in the interrupt mode, servicing an interrupt for each byte consumes a significant amount of time, especially if bytes come back to back. Polling provides a solution to this problem. Therefore, the combination of polls and interrupts creates a very efficient mode.

The 8044 side of the interface supports this mode as well.



NOTES:
1. Optional path. Polling may be done multiple times before return.
2. This flow chart applied to both sides of the interface. Parameters in parentheses refer to the 8044 side.
3. Interrupts are assumed disabled when entering interrupt routine.

1786

Figure 12-6. Polled/Interrupt Mode Send

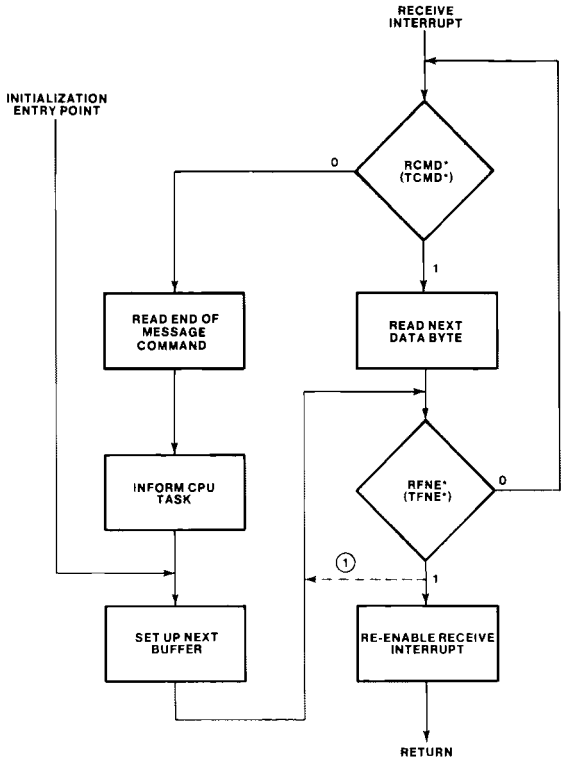


Figure 12-7. Polled/Interrupt Mode Receive

12.2.2.4 DMA Operation

The iSBX 344 board supports a limited DMA interface to the extension device only. The limitation in the interface is the absence of DMA acknowledge signals. As a result, the interface does not meet iSBX bus DMA specification and will not work with 8237-type DMA controllers. The main purpose for providing this feature is for use with the 80186/188 DMA controller.

The DMA interface uses three signals to connect the extension device: Transmit DMA Request (TDRQ), Receive DMA Request (RDRQ), and Receive Command Interrupt (RCMI).

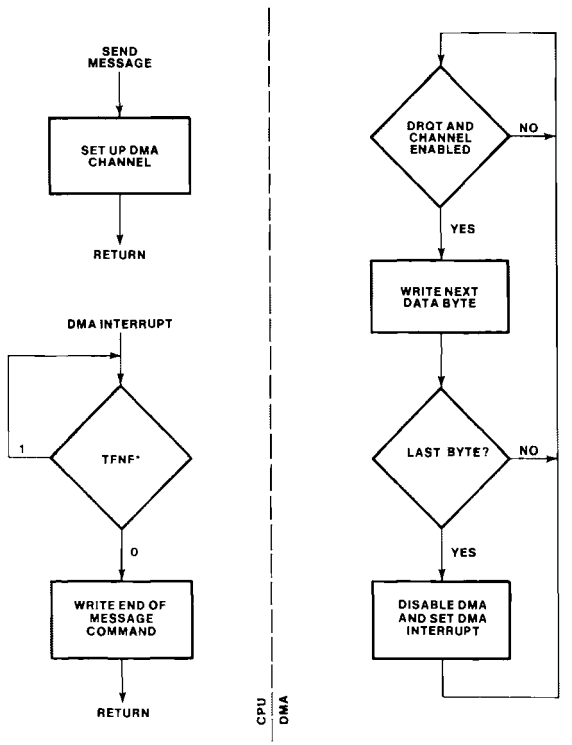
TDRQ — The TDRQ signal becomes active whenever a byte can be written into the transmit FIFO. This signal is logically equivalent to the TFNF* bit in the status register.

RDRQ — The RDRQ signal becomes active whenever a data byte is valid in the receive buffer. This signal is inactive when a command is valid in the receive FIFO.

RCMI — The RCMI interrupt signal becomes active only when a command is valid in the receive FIFO.

Figure 12-8 shows a the sequence of a typical DMA mode message send. To initiate the send, the extension device sets up a DMA channel to transmit to the data port. The DMA controller then transfers the message without further CPU intervention. Upon completion of the DMA, the DMA controller interrupts the CPU. The CPU then sends the end of message command as it did in the other modes.

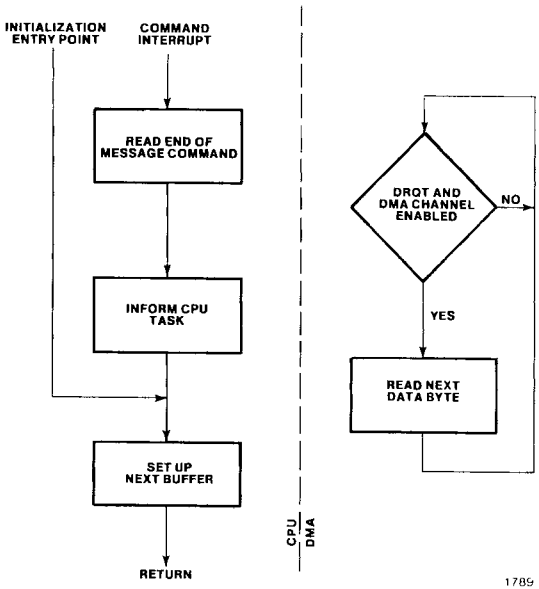
Figure 12-9 shows a typical DMA mode receive message. Upon initialization, the extension device sets up a receive buffer of maximum message length. When a message arrives, it is transferred to the specified buffer without CPU intervention. When the end of message command arrives, it interrupts the CPU or the extension device which terminates DMA and sends the EOM command. The extension device then reads the end of message command and sets up another buffer. This scheme allows messages from 1 to n bytes to be transferred into an n byte buffer.



NOTES: 1. This flow chart applies only to the extension side of the interface.

1788

Figure 12-8. DMA Mode Send



NOTES: 1. This flow chart applies only to the extension side of the interface.

Figure 12-9. DMA Mode Receive

12.3 EQUIPMENT SUPPLIED & EQUIPMENT REQUIRED

The iSBX 344 board includes a schematic diagram of the board shipped in the same container as the board:

In a typical system installation, you will need, at a minimum, the following equipment:

- Connectors - 10 pin plug, flat cable/discrete wire
- BITBUS Cables
- iSBC or iRCB baseboard

12.4 SPECIFICATIONS

Table 12-1 contains a list of specifications for the iSBX 344 board.

Table 12-1. Specifications

CPU	8044 Microcontroller
WORD SIZE	
Instruction	8 bits
Data	8 bits
PROCESSOR CLOCK SPEED	12.0 MHz
INSTRUCTION EXECUTION TIME	1 usec 60% instructions 2 usec 40% instructions 4 usec Multiply & Divide
EXTERNAL MEMORY	
Data Memory Site	Accepts 2K x 8 through 32K x 8 SRAMs
Addresses	Option A - 0000H-7FFFH Option B - 0000H-7FFFH
Code Memory Site	Accepts various sizes of ROM, EPROM, SRAM, NVRAM, and EEPROM devices
Addresses	Option A - 1000H-0FFFFH (0000H-FFFFH if EA* is active) Option B - 8000H-0FEFFH
EXTERNAL I/O SPACE	0FF00H-0FFFFH (mapped into the data memory space)

Table 12-1. Specifications (continued)

PHYSICAL CHARACTERISTICS	Double-wide iSBX MULTIMODULE Form Factor
Dimensions	
Width	63.5 mm (2.50 in)
Height	10.16 mm (0.4 in) maximum component height
Depth	190.5 mm (7.5 in)
Weight	113 gm (4 ounces)
ENVIRONMENTAL REQUIREMENTS	
Operating Temperature	0°C to 55°C at 200 Linear Feet/Minute (LFM) air velocity
Relative Humidity	90% Non-condensing
Storage Temperature	-40°C to 75°C
Storage Humidity	90% Non-condensing
INTERFACES	
iSBX I/O Expansion Bus	Compliance Level - D8
BITBUS Interconnect	Fully supports both synchronous and self-clocked modes for 375kb/sec and 62.5kb/sec bit rates
POWER REQUIREMENTS	0.9A at +5V $\pm$ 5% (memory not included)

12.4.1 COMPLIANCE LEVEL iSBX™ BUS SPECIFICATION

All Intel iSBX Bus-compatible products are designed around guidelines set forth in the Intel iSBX BUS SPECIFICATION. The specification requires that certain board operating characteristics, such as the data bus width, be clearly stated in the board's printed specifications. This information quickly summarizes the level of compliance the board bears to the published iSBX bus specification. It clearly states the board's level of compatibility to the iSBX Bus structure. Intel's iSBX BUS SPECIFICATION contains more information.

The following notation states the iSBC 344 board's level of compliance to the iSBX BUS SPECIFICATION.

D8

Where: D8 = 8-bit expansion module.

12.4.2 BITBUS™ INTERCONNECT SUPPORT

The iSBX 344 board supports both synchronous and self-clocked modes of BITBUS operation. This interface is electrically RS485 compatible. Connector information is located in the connector section of this chapter. Refer to the BITBUS section of Intel's DISTRIBUTED CONTROL MODULES DATA BOOK for details.

12.4.2 CONNECTOR INFORMATION

The iSBX 344 board contains an iSBX connector (P1) and a serial I/O connector (J1). Figure 12-10 shows the locations of these connectors on the iSBX 344 board.

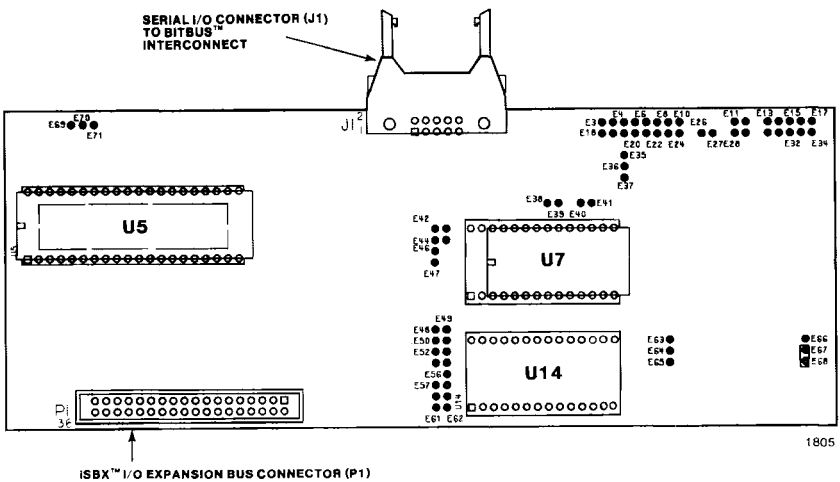


Figure 12-10. Connector Location Diagram

Refer to the Intel iSBX BUS SPECIFICATION for a description of the pin assignments, AC characteristics, and DC characteristics of the P1 connector interface on the iSBX 344 board.

Table 12-2 lists the pin-out for the iSBX 344 board's serial I/O connector (J2). Table 12-3 provides the numbers for specific vendor parts that connect to the J2 connector.

Table 12-2. J2 Connector Pin-Out

Pin	Signal	Pin	Signal
1	+12V	6	DATA
2	+12V	7	DCLK*/RTS*
3	GND	8	DCLK/RTS
4	GND	9	RGND
5	DATA*	10	RGND

Table 12-3. Serial Connector Options

Function	Vendor Part Number
Plug-Flat Cable	3M 3473-6010, TB Ansley 609-1001M, or equal
Plug-Discrete Wire	Berg 65846-007, ITT Cannon 121-7326-105, or equal

12.5 CONFIGURATION

This section explains the jumper options on the iSBX 344 BITBUS Controller MULTIMODULE board and describes how to configure the jumpers for specific applications. The user should read this entire section before configuring the jumpers on the board.

12.5.1 CONFIGURATION OVERVIEW

The configuration options for the iSBX 344 board appear in the following functional groups:

- Address and Mode Configuration
- Memory Configuration
- Serial Interface Configuration

Some of these configuration options are interrelated. Selecting one option may require reconfiguring another. Therefore, the user should consider the entire application before configuring the board.

12.5.2 JUMPER CONFIGURATIONS

The following sections describe the jumper configuration options on the iSBX 344 board. Section 12.5.2.5 contains a jumper location diagram for use during the configuration process.

12.5.2.1 Default Jumper Configurations

The iSBX 344 BITBUS Controller MULTIMODULE board leaves the factory in a specific configuration called the default configuration. Section 12.5.2.5 contains a list of all default jumpers. In addition, a "§" symbol calls out any jumper connection which is a default connection. The jumper configuration will have to be changed to operate in your particular application.

12.5.2.2 Address and Mode Configuration

The iSBX 344 board provides two registers on I/O ports which the DCM Controller reads for configuration information. These registers are the mode register and the node address register. Some of the bits in these registers are also used to perform functions on the board. Figure 12-11 shows the location of these jumpers on the iSBX 344 board.

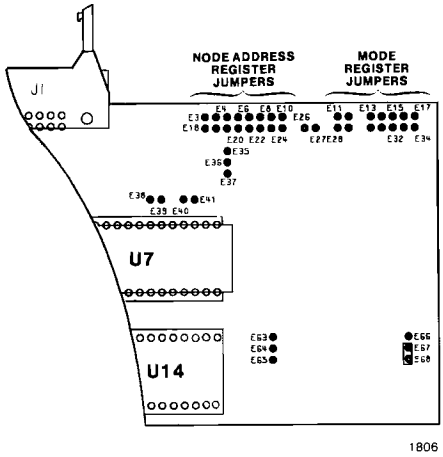


Figure 12-11. Address And Mode Jumper Locations

12.5.2.2.1 NODE ADDRESS REGISTER. The DCM Controller uses the 8-bit value in the node address register for its node address on power-up or reset. All jumpers removed corresponds to node 00H, and all jumpers installed corresponds to node address FFH. The default configuration is node address 00H (all jumpers removed). Figure 12-12 summarizes the node register jumpers.

CORRESPONDING JUMPER PAIR	E3 E18	E4 E19	E5 E20	E6 E21	E7 E22	E8 E23	E9 E24	E10 E25
BIT	7	6	5	4	3	2	1	0

1807

Figure 12-12. Node Address Register

12.5.2.2.2 MODE REGISTER. The DCM reads the mode register on power-up and reset to obtain configuration parameters. Figure 12-13 summarizes the mode register and shows the corresponding jumper numbers. The default configuration is all jumpers removed (register contents = 00).

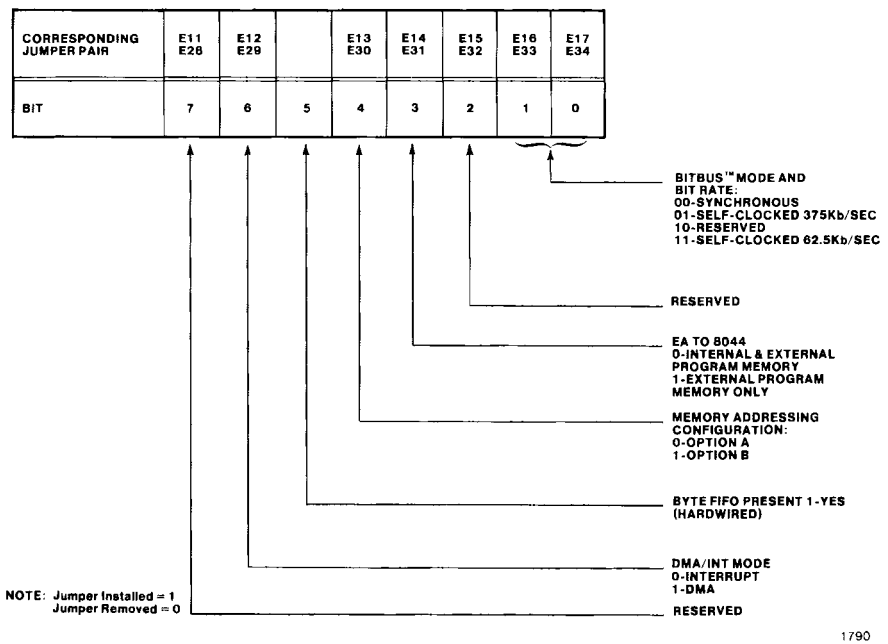


Figure 12-13. Mode Register

12.5.2.3 Memory Configuration

The iSBX 344 board supports a wide range of byte-wide memory devices and offers two memory addressing options. Section 11.3.4.2 of this manual contains the information on the devices that are compatible with the DCM core (and therefore compatible with the iSBX 344 MULTIMODULE board). That section also discusses the AC and DC specifications of the byte-wide sockets and details the universal memory site jumper matrix configuration. This section discusses the location of the memory configuration jumpers on the iSBX 344 board and the default configuration of these jumpers.

Figure 12-14 shows the location of the memory site configuration jumpers. The jumper matrices are shown in their default configuration. The iSBX 344 board is shipped with a 2k x 8 RAM device in the data site. The default configuration for the code site is for a 16k x 8 EPROM device.

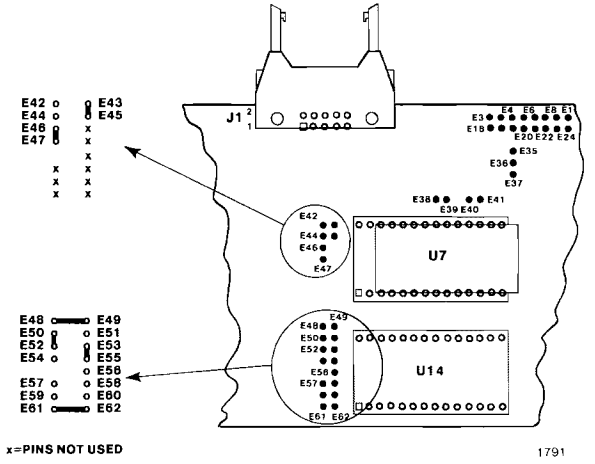


Figure 12-14. Memory Site Configuration Jumpers Location

12.5.2.4 Serial Interface Configuration

The serial port interface may be configured for either synchronous mode or self-clocked mode operation. In the synchronous mode, the clock source is also selectable. Table 12-4 summarizes these options.

Table 12-4. Serial Interface Options

Option	Jumpers	
	Install	Remove
Synchronous Mode	E35-E36§, E38-E39§, E40-E41§, and E69-E70	E70-E71§
Self-clocked Mode	E36-E37 and E70-E71§	E38-E39, E40-E41, and E69-E70
ALE to Clock	E66-E67	E67-E68§
Oscillator to Clock	E67-E68§	E26-E27
MCLK to Clock	E26-E27 and E67-E68§	E66-E67

12.5.2.4.1 TERMINATION RESISTORS. In either mode of operation, the last board on each end of the BITBUS interconnect must provide termination resistors. The other boards in the system must not have termination resistors. The iSBX 344 board provides sockets for these resistors. The resistors should be 1/4 watt. The value of the resistors should match the characteristic impedance of the cable as closely as possible, with the minimum value being 120 ohms. The socket locations are as follows:

Socket	Function
R10	DATA signals termination
R13	DCLK/RTS signals termination

R13 is only necessary for synchronous mode and self-clocked mode when repeaters are present.

12.5.2.4.2 SERIAL CLOCK OPTIONS. The iSBX 344 board has offers several options for producing the SCLK signal. In synchronous mode using the oscillator for the clock source, an oscillator must be installed in Y2. Table 12-4 lists the jumpers associated with this option. The following formula determines the oscillator-produced SCLK frequency:

$$SCLK = \frac{\text{Oscillator frequency}}{4}$$

The ALE signal can also be used for the serial clock. The frequency of the ALE signal is approximately 2MHz. By installing jumper E66-E67, you route the ALE signal through the flip-flop, divide-by-two, and produce an SCLK of approximately 1MHz. Refer to Chapter 11 for more details of the ALE option.

In addition, if the MCLK signal is less than or equal to 9.6MHz, it can be used to produce SCLK. Table 12-4 lists the associated jumpers. The following equation determines the frequency of the MCLK-produced SCLK:

$$SCLK = \frac{\text{MCLK frequency}}{4}$$

12.5.2.5 Numerical List of Jumpers

This section provides an overview of the jumpers on the iSBX 344 board. Figure 12-15 shows the approximate location of each jumper on the iSBX 344 board. Table 12-5 lists all the jumpers and their functions and Table 12-6 lists all of the jumpers that are installed in the default configuration of the board.

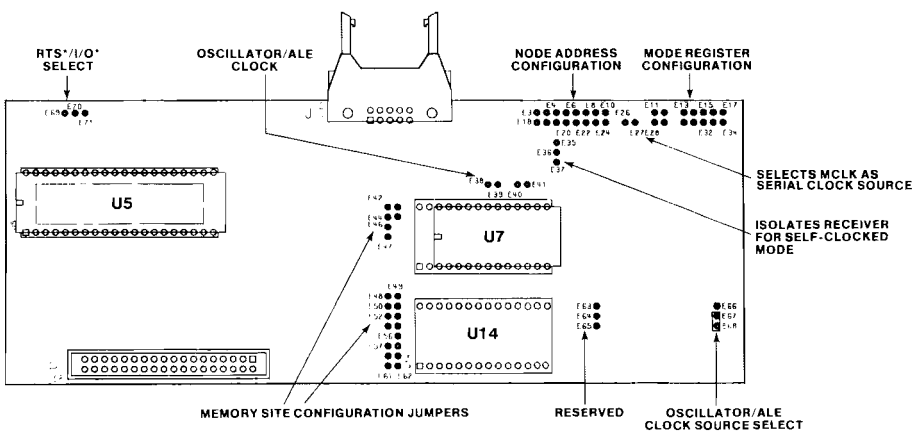


Figure 12-15. Jumper Location Diagram

Table 12-5. Numerical List Of Jumpers And Their Functions

Jumper Number	Functions
E3-E18	Sets bit 7 of node address register
E4-E19	Sets bit 6 of node address register
E5-E20	Sets bit 5 of node address register
E6-E21	Sets bit 4 of node address register
E7-E22	Sets bit 3 of node address register
E8-E23	Sets bit 2 of node address register
E9-E24	Sets bit 1 of node address register
E10-E25	Sets bit 0 of node address register
E11-E28	Reserved
E12-E29	DMA/INT mode select
E13-E30	Memory addressing configuration
E14-E31	EA to 8044
E15-E32	Reserved
E16-E33 and E17-E34	Set BITBUS mode and bit rate
E26-E27	Selects MCLK as serial clock source
E35-E36§	Connects SCLK to processor (synch. mode)
E36-E37	Ties RTS low for RTS output (self-clocked mode)
E38-E39§	Removes SCLK receiver load for self-clocked mode
E40-E41§	Removes SCLK receiver load for self-clocked mode
E42 thru E47	Data site memory configuration E43-E45§ E46-E47§
E48 thru E62	Code site memory configuration E48-E49§, E50-E52§ E53-E55§, E61-E62§
E63-E64	Not used
E64-E65	Not used
E66-E67	Selects ALE as clock source
E67-E68§	Selects Oscillator clock or MCLK as clock source
E69-E70	Selects RTS* for transceiver direction (synch. mode)
E70-E71§	Selects I/O for transceiver direction (self-clocked mode)

Table 12-6. Default Jumpers (As-Shipped Configuration)

Jumper Connections		
E35-E36§ E43-E45§ E50-E52§ E67-E68§	E38-E39§ E46-E47§ E53-E55§ E70-E71§	E40-E41§ E48-E49§ E61-E62§

12.6 PROGRAMMING CONSIDERATIONS

The iSBX 344 board has two programming interfaces: the DCM Controller and the byte FIFO.

12.6.1 DCM CONTROLLER PROGRAMMING

Programming DCM Controller with user tasks is optional. The firmware provided already handles communications and I/O functions. If you desire additional capabilities, you can add user tasks. Refer to Chapter 5 for information on task generation. You must write these tasks in either ASM 51 or PL/M 51 and store them in a memory device in the code memory site as iRMX 51 tasks.

When adding user tasks to the iSBX 344, it is usually not necessary to know any I/O addresses because the firmware handles both the parallel and serial interfaces. For special cases where a user task needs direct access to an I/O function, Table 12-7 lists the addresses. These addresses are either byte-addressed or bit-addressed memory as indicated in the table.

Table 12-7. DCM Controller (8044) I/O Addressing On iSBX™ 344 Board

Function	Address	Read	Write	Bit
Red LED	90H	X	X	X
Green LED	91H	X	X	X
Status				
TCMD*	92H	X		X
RFNF*	B3H	X		X
TFNE*	B2H	X		X
RDY/NE*	B4H	X	X	X
Data	FF00H	X	X	
Command	FF01H	X	X	
Node Address	FFFFH	X		
Configuration	FFFEH	X		

12.6.2 BYTE FIFO PROGRAMMING

Some kind of software driver is needed to control the byte FIFO interface on the extension device. The iRMX 510 support package provides drivers for applications using the iRMX 86 Operating System, iRMX 286R Operating System, iRMX 88 Executive, or the ISIS-based iPDS Personal Development System. For other applications, the user must provide a driver modeled after one of the examples that Section 12.2.2 presents. For these cases, Table 12-8 lists the iSBX 344 board addressing.

Table 12-8. iSBX™ 344 Board Addressing

Register Function	Address		Comments
	8-Bit Mode	16-Bit Mode	
DATA	BASE	BASE	Read/Write
COMMAND	BASE+1	BASE+2	Write sets command from extension. Read clears command to extension.
STATUS	BASE+2	BASE+4	Read Only

Note: MCS0* address range on extension device determines BASE. Refer to the hardware reference manual of extension device for this address range.

Figure 12-15 defines the status register of the byte FIFO. Table 12-9 presents the iSBX 344 board signals and the corresponding iSBX I/O expansion bus signals for the interrupt and DMA options.

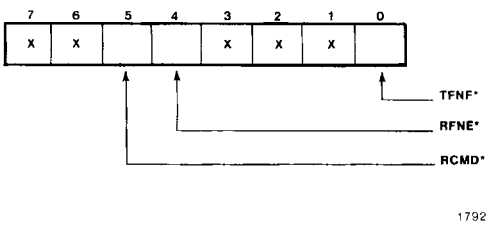


Figure 12-16. iSBX™ 344 Status Register

Table 12-9. iSBX™ 344 Interrupt And DMA Signals

Interface Option	iSBX™ 344 Signal	iSBX™ Bus Signals
INT	RINT	MDRQT/MINTR0
INT	TINT	MINTR1
INT OR DMA	RCMI	OPT0
DMA	RDRQ	MDRQT/MINTR0
DMA	TDRQ	MINTR1

***

CHAPTER 13
USING THE iRCB 44/10 REMOTE CONTROLLER BOARD

13.1 INTRODUCTION

The iRCB 44/10 Remote Controller Board is a DCM core-based single board computer with an iSBX bus connector for I/O expansion and a DIN (Deutches Institute fur Normung) connector with parallel I/O. The iRCB 44/10 board is part of the Distributed Control Modules product line, providing a low-cost master or slave BITBUS node with I/O capability. Figure 13-1 shows the iRCB 44/10 board. The board contains a DIN connector with 24 lines of parallel I/O (16 parallel I/O and 8 parallel input), an iSBX connector, and a serial I/O connector.

This chapter provides the information that is necessary for you to install and configure a iRCB 44/10 board in a BITBUS system. This chapter contains a high-level description of the iRCB 44/10 board, a list of the board's specifications, and a discussion of the configuration options available on the board.

Because the iRCB 44/10 board is based on the DCM core, many of the discussions in this chapter refer you to Chapter 11 which provides a more in-depth operational discussion of the core. In order to avoid redundancy, this chapter is limited to configuration information and operational discussions of only the sections of the iRCB 44/10 board that are not part of the core.

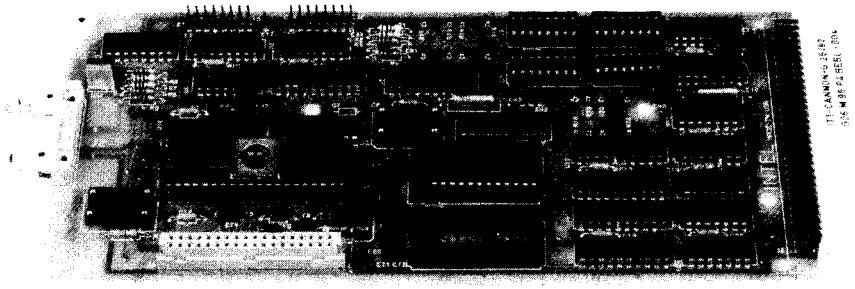


Figure 13-1. iRCB 44/10 Remote Controller Board

13.2 DESCRIPTION

This section provides a brief discussion of the key features of the iRCB 44/10 Remote Controller Board. Figure 13-2 provides a functional block diagram of the iRCB 44/10 board. As the figure shows, the iRCB 44/10 board consists of the DCM core enhanced with an iSBX I/O Expansion bus connector and 24 lines of parallel I/O. Chapter 11 of this manual contains a detailed discussion of the DCM core; therefore, this section only highlights the major features of the core. In addition, this section discusses the parallel I/O and the iSBX I/O Expansion.

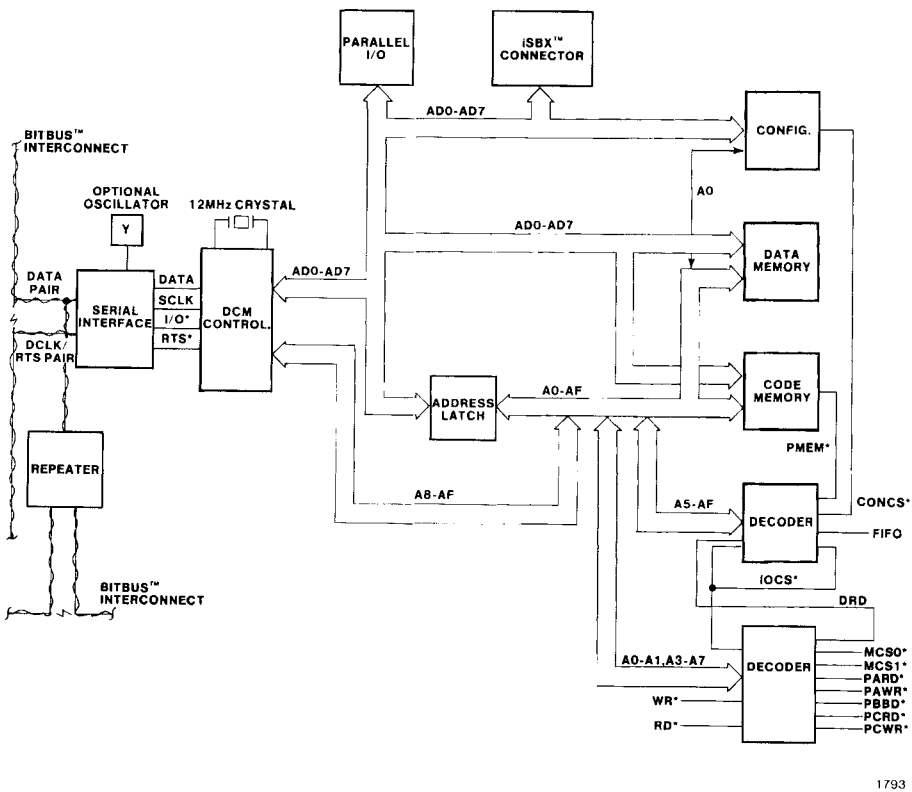


Figure 13-2. iRCB 44/10 Block Diagram

13.2.1 THE DCM CORE

The DCM core is the combination of hardware and software elements that are common to all boards in the DCM product line. The DCM core provides the foundation for the iRCB 44/10 Remote Controller Board. As the areas outlined in red in Figure 13-2 illustrate, the core consists of the DCM Controller, an address latch, a decoder, data memory space, code memory space, a serial interface, and a configuration section. The code and data memory spaces consist of two 28-pin sites. One site is dedicated to RAM for data memory while the other site is dedicated to program store.

13.2.2 PARALLEL I/O

The iRCB 44/10 board provides 24 lines of parallel I/O on the P1 connector. Sixteen of the I/O lines are fully programmable as inputs or outputs, with loopback on a bit-by-bit basis. The remaining eight are dedicated as inputs. This parallel port structure is fully programmable in software. It requires no jumpers for configuration.

Figure 13-3 shows the structure of a single I/O bit. This figure represents the bit structure of I/O ports A and C. The bits of port B, the 8-bit input only port, do not have the a latch or open collector driver. Upon power-up, the system sets the bits of Ports A and C to input mode by resetting the latch. This disables the open collector output driver. After power-up you can use the port as an input, by reading it, or an output, by writing to it. The loopback capability provides a diagnostic tool as well as the ability to easily perform bit set, reset, and toggle operations. The I/O programming section of this chapter provides further details on these capabilities.

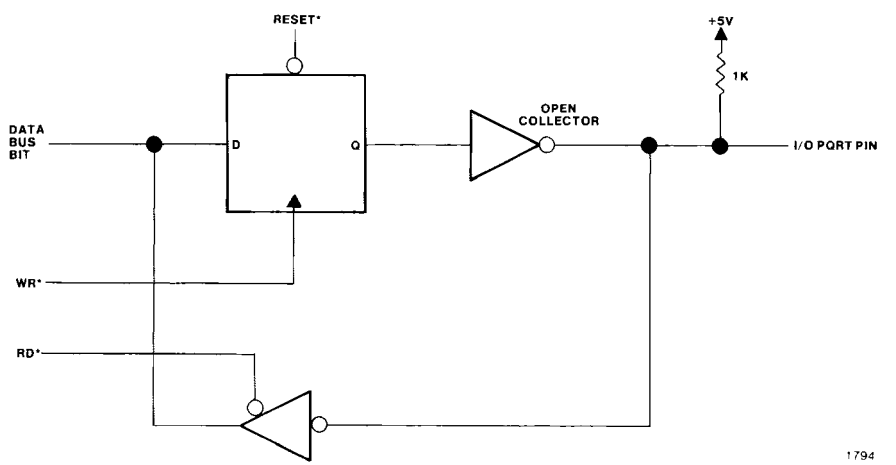


Figure 13-3. I/O Port Structure

13.2.3 iSBX™ I/O EXPANSION

The iRCB 44/10 board provides one iSBX I/O Expansion Bus connector (J1) that accepts the complete line of full-speed 8-bit I/O expansion modules. The only incompatible modules are those that require the MWAIT* signal or DMA operation. Possible compatible modules include parallel I/O, serial I/O, BITBUS expansion, analog input, analog output, and IEEE 488.

13.2.4 BITBUS™ REPEATER

The iRCB 44/10 board provides the capability for an on-board BITBUS repeater. By adding repeaters to your BITBUS system, you can increase the number of nodes and the distance of the system. The iRCB 44/10 board contains sockets for user-supplied resistors and transmitter/receiver components.

Repeaters for the BITBUS are relatively simple. A BITBUS repeater may or may not provide electrical isolation, depending on the application requirements. When a slave is not transmitting, all RTS signals are high impedance. Biasing resistors on the serial link hold the line in the inactive state. This in turn points all data line buffers away from the master. When the slave transmits, it activates RTS which turns around all repeaters between it and the master node. Figure 13-4 shows the repeater configuration used in the DCM core.

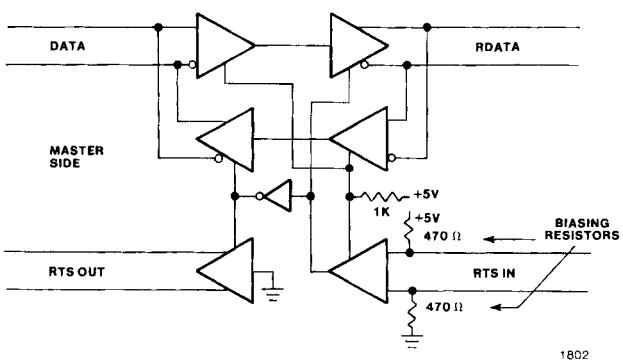


Figure 13-4. DCM Core Repeater Configuration

13.3 EQUIPMENT SUPPLIED AND EQUIPMENT REQUIRED

The iRCB 44/10 board includes a schematic diagram of the board shipped in the same container as the board.

In a typical system installation, you will need, at a minimum, the following equipment:

- Connectors - 10 pin plug, flat cable/discrete wire
  - 64 pin DIN connector
- Cables

13.4 SPECIFICATIONS

This section outlines the specifications for the iRCB 44/10 board and discusses interface compliance levels and connector information. Table 13-1 contains a list of specifications for the iRCB 44/10 board.

Table 13-1. Specifications

CPU	8044 Microcontroller
WORD SIZE	
Instruction	8 bits
Data	8 bits
PROCESSOR CLOCK SPEED	12.0 MHz
INSTRUCTION EXECUTION TIME	1 usec 60% instructions 2 usec 40% instructions 4 usec Multiply & Divide
EXTERNAL MEMORY	
Data Memory Site	Accepts 2K x 8 through 32K x 8 SRAMs
Addresses	Option A - 0000H-7FFFH Option B - 0000H-7FFFH
Code Memory Site	Accepts various sizes of ROM, EPROM, SRAM, NVRAM, and EEPROM devices.
Addresses	Option A - 1000H-FFFFH (0000H-FFFFH if EA* is active) Option B - 8000H-FEFFH
EXTERNAL I/O SPACE	FF00H-FFFFH (mapped into the data memory space)

Table 13-1. Specifications (continued)

PHYSICAL CHARACTERISTICS	
Width	13.77 mm (.542 in) maximum component height
Height	100mm (3.93 in)
Depth	220mm (8.65 in)
Weight	169 gm (6 ounces)
ENVIRONMENTAL REQUIREMENTS	
Operating Temperature	0°C to 55°C at 200 Linear Feet/Minute Air Velocity
Relative Humidity	90% Non-condensing
Storage Temperature	-40°C to 75°C
Storage Humidity	90% Non-condensing
INTERFACES	
iSBX I/O Expansion Bus	Compliance Level - D8/8F
BITBUS™ Interconnect	Fully supports both synchronous and self-clocked modes for 375kb/sec and 62.5kb/sec bit rates
POWER REQUIREMENTS	0.9A at +5V ± 5% (memory, repeater, and iSBX board not included)

13.4.1 COMPLIANCE LEVEL: iSBX™ BUS SPECIFICATION

All Intel iSBX Bus-compatible boards are designed around guidelines set forth in the Intel iSBX I/O EXPANSION BUS SPECIFICATION. The specification requires that certain board operating characteristics, such as the data bus width, be clearly stated in the board's printed specifications. This information quickly summarizes the level of compliance the board bears to the published iSBX Bus Specification. It clearly states the board's level of compatabilty to the iSBX Bus structure. The Intel iSBX I/O EXPANSION BUS SPECIFICATION contains more information.

The following notation states the iRCB 44/10 board's level of compliance to the Intel iSBX I/O EXPANSION BUS SPECIFICATION.

D8/8F

Where:

D8/8 = interfaces to an 8-bit baseboard that can support an 8-bit expansion module.

F = does not support interlocked operation. Only full-speed operations are performed.

13.4.2 BITBUS™ INTERCONNECT SUPPORT

The iRCB 44/10 board supports both synchronous and self-clocked modes of BITBUS Interconnect operation. This interface is electrically RS485 compatible. Connector information is located in the connector section of this chapter. Refer to the BITBUS interconnect section of Intel's DISTRIBUTED CONTROL MODULES DATA BOOK for details.

13.4.3 CONNECTOR INFORMATION

The iRCB 44/10 board contains an DIN connector (P1) with 24 lines of parallel I/O and the BITBUS interface, an iSBX connector (J1), and a serial I/O connector (J2) with the BITBUS repeater interface. Figure 13-5 shows the locations of these connectors on the iRCB 44/10 board.

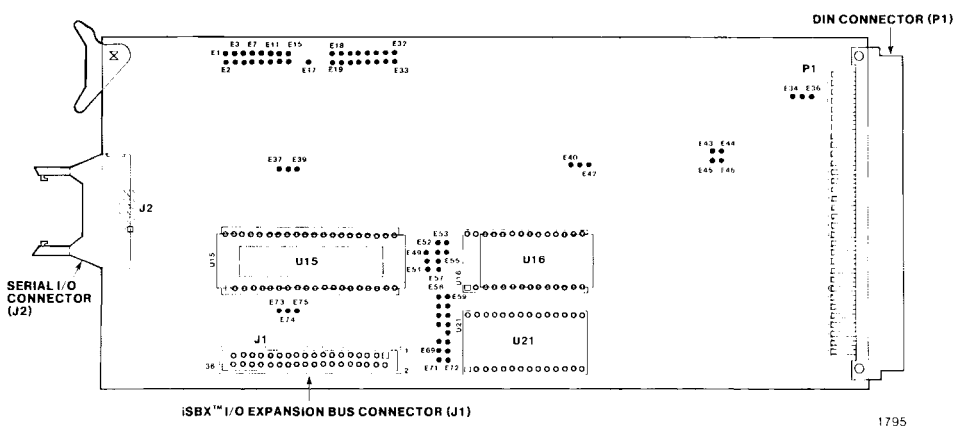


Figure 13-5. Connector Location Diagram

The P1 connector contains both the parallel I/O and the BITBUS interface. Table 13-2 shows the pin-out for the P1 connector, and Table 13-3 lists the connector's electrical specifications for the I/O lines. PA0-PA7, PB0-PB7, and PC0-PC7 are the bits of the three 8-bit parallel ports on the iRCB 44/10 board. The DIN pin numbers are for the 64-pin connector on the board. The pin & socket pin numbers are for the 50-pin connector that attaches to the I/O conditioning strip.

Table 13-2. iRCB 44/10 P1 Connector Pin-out

DIN Pin #	Pin & Socket Pin #	Function	DIN Pin #	Pin & Socket Pin #	Function
1a		GND	1c		GND
2a		+5V	2c		+5V
3a		DATA	3c		DATA*
4a		DCLK/RTS	4c		DCLK*/RTS*
5a	1	EXT INT	5c		RGND
6a	3	PB7	6c	2	GND
7a	5	PB6	7c	4	GND
8a	7	PB5	8c	6	GND
9a	9	PB4	9c	8	GND
10a	11	PB3	10c	10	GND
11a	13	PB2	11c	12	GND
12a	15	PB1	12c	14	GND
13a	17	PB0	13c	16	GND
14a	19	PC3	14c	18	GND
15a	21	PC2	15c	20	GND
16a	23	PC1	16c	22	GND
17a	25	PC0	17c	24	GND
18a	27	PC4	18c	26	GND
19a	29	PC5	19c	28	GND
20a	31	PC6	20c	30	GND
21a	33	PC7	21c	32	GND
22a	35	PA7	22c	34	GND
23a	37	PA6	23c	36	GND
24a	39	PA5	24c	38	GND
25a	41	PA4	25c	40	GND
26a	43	PA3	26c	42	GND
27a	45	PA2	27c	44	GND
28a	47	PA1	28c	46	GND
29a	49	PA0	29c	48	GND
30a		+12V	30c		-12V
31a		+5V	31c		+5V
32a		GND	32c		GND

Note: Pin 50 of the pin & socket connector can be +5V, -12V, or NC, depending upon the requirements of your application.

Table 13-3. Parallel I/O Electrical Specification

Parameter	Condition	Minimum	Maximum	Units
V_{OL}	I_{OL}=16mA		0.5	V
V_{OH}	I_{OH}=-2mA	2.4		V
V_{IH}		2.0	7.0	V
V_{IL}		-1.0	0.8	V
I_{IL}	V_{IL}=0.5V		-6.0	mA
I_{IH}	V_{IH}= logic high		0	mA
I_I	V_{IH}=7V		-3.0	mA

Table 13-4 lists the pin-out for the repeater connector (J2), which is the same as the pin-out for the iSBX 344 board's serial I/O connector. Table 13-5 provides the numbers for specific vendor parts that connect to the J2 connector. Table 13-6 lists some specific vendor parts that connect to the P1 connector.

Refer to the Intel iSBX I/O EXPANSION BUS SPECIFICATION for a description of the pin assignments, AC characteristics, and DC characteristics of the J1 connector interface on the iRCB 44/10 board.

Table 13-4. J2 Connector Pin-out

Pin	Signal	Pin	Signal
1	+12V	6	DATA
2	+12V	7	DCLK*/RTS*
3	GND	8	DCLK/RTS
4	GND	9	RGND
5	DATA*	10	RGND

Table 13-5. Serial Connector Options

Function	Vendor Part Number
Plug-Flat Cable	3M 3473-6010, TB Ansley 609-1001M, or equal
Plug-Discrete Wire	Berg 65846-007, ITT Cannon 121-7326-105, or equal

Table 13-6. DIN Connector Options

Function	Vendor Part Number
Plug-Flat Cable	GW Elco 00-8259-096-84-124, Robinson Nugent RNE-IDC64C-TG30, or equal
Plug-Discrete Wire	ITT Cannon G06 M96 P3BDBL-004, GW Elco 60 8257 3017, or equal

13.5 CONFIGURATION

This section explains the jumper options on the iRCB 44/10 Remote Controller Board and describes how to configure the jumpers for given applications. You should read this entire section before configuring the jumpers on the board.

13.5.1 CONFIGURATION OVERVIEW

The configuration options for the iRCB 44/10 board fall into the following functional groups.

- Address and Mode
- Memory Configurator
- Serial Interface
- Repeater Interface
- Interrupt Sources

Some of these configuration options are interrelated. Selecting one option might require reconfiguring another. Therefore, you should consider the entire application before configuring the board.

13.5.2 JUMPER CONFIGURATIONS

The following sections describe the jumper configuration options on the iRCB 44/10 board. When configuring your board, refer to the jumper location diagram in section 13.5.2.7.

13.5.2.1 Default Jumper Configurations

The iRCB 44/10 Remote Controller Board leaves the factory in a specific configuration called the default configuration. In addition, any jumper connection listed in this chapter which is a default connection is called out with a "§" symbol. Section 13.5.2.7 of this chapter contains a list of all default jumpers. The jumper configuration will have to be changed to operate in your particular application.

13.5.2.2 Address and Mode Configuration

The iRCB 44/10 board provides two jumper-configurable registers, the node address register and the mode register, on I/O ports which the DCM Controller may read on power-up for configuration information. Some of the bits in these registers are also used to perform functions on the board. Figure 13-6 shows the location of these jumpers on the iRCB 44/10 board. For more information on the functions of the node address and mode registers, refer to Section 11.3.6 of this manual.

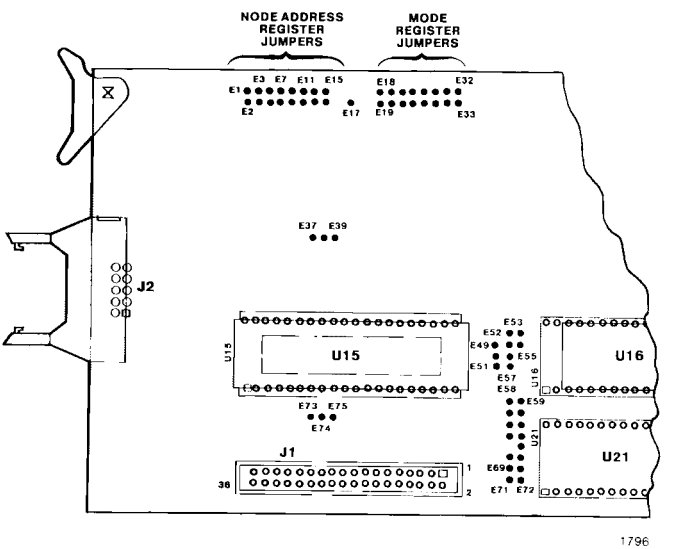


Figure 13-6. Address And Mode Jumper Locations

13.5.2.2.1 NODE ADDRESS REGISTER. The node address register provides an 8-bit node address that the DCM Controller uses on power-up and reset. All jumpers removed corresponds to node 0 while all jumpers installed to node address FFH. Default configuration is node address 0 (all jumpers removed). Figure 13-7 shows the jumpers associated with the node address register.

CORRESPONDING JUMPER PAIR	E1 E2	E3 E4	E5 E6	E7 E8	E9 E10	E11 E12	E13 E14	E15 E16
BIT	7	6	5	4	3	2	1	0

1797

Figure 13-7. Node Address Register

13.5.2.2.2 MODE REGISTER. The DCM reads the mode register on power-up and reset to obtain configuration parameters. Figure 13-8 depicts the mode register and lists the associated jumpers and their functions. Default configuration is all jumpers removed (register contents = 00).

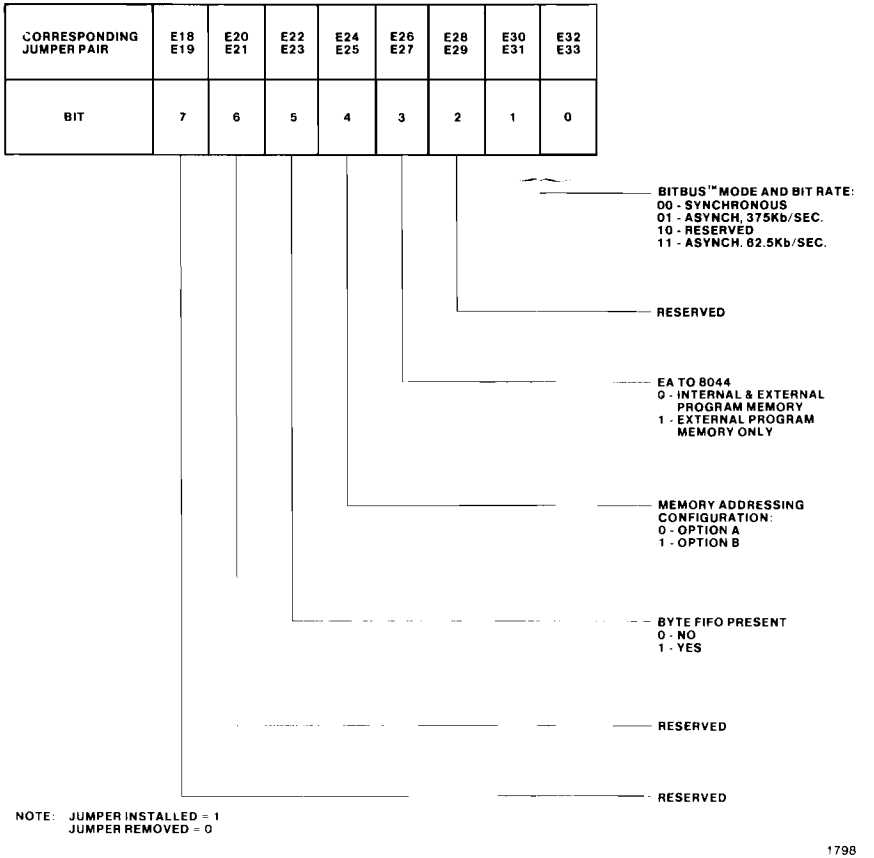


Figure 13-8. Mode Register

13.5.2.3 Memory Configuration

The iRCB 44/10 board supports a wide range of byte-wide memory devices and offers two memory addressing options. Section 11.3.4.2 of this manual contains the information on the devices that are compatible with the DCM core (and therefore compatible with the iRCB 44/10 board). That section also discusses the AC and DC specifications of the byte-wide sockets and details the universal memory site jumper matrix configuration. This section discusses the location of the memory configuration jumpers on the iRCB 44/10 board and the default configuration of these jumpers.

Figure 13-9 shows the location of the memory site configuration jumpers. The jumper matrices are shown in their default configuration. The iSBX 344 board is shipped with a 2k x 8 RAM device in the data site. The default configuration for the code site is for a 16k x 8 EPROM device.

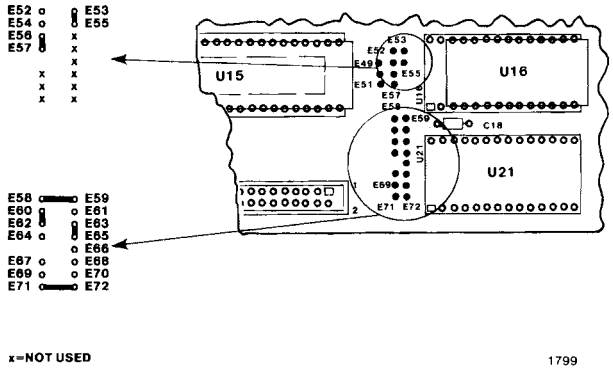


Figure 13-9. Memory Site Configuration Jumpers Location

13.5.2.4 Serial Interface Configuration

You can configure the serial port interface for either synchronous mode or self-clocked mode operation. In the synchronous mode, the clock source is also selectable. Table 13-7 summarizes these options.

Table 13-7. Serial Interface Options

Option	Jumpers	
	Install	Remove
Synchronous Mode	E34-E35§, E43-E44§, E45-E46§, and E73-E74	- E74-E75§
Self-clocked Mode	E35-E36 and E74-E75§	E43-E44, E45-E46, and E73-E74
ALE Clock	E38-E39	E37-E38§
Y1 Oscillator Clock	E37-E38§	-
Y2 Oscillator Clock	-	E37-E38§ and E38-E39

In either mode of operation, the last board on each end of the BITBUS interconnect must provide termination resistors. The other boards in the system must not have termination resistors. The iRCB 44/10 board provides sockets for these resistors. The resistors should be 1/4 watt. The value of the resistors should match the characteristic impedance of the cable as closely as possible, with the minimum value being 120 ohms. Table 13-8 specifies the socket locations and their function. Figure 13-10 shows where these user provided components reside on the iRCB 44/10 board.

Table 13-8. Termination Resistor Sockets

Socket	Function
R19	DCLK/RTS signals termination
R20	DATA signals termination

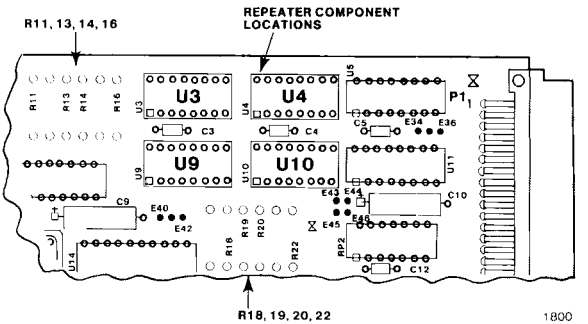


Figure 13-10. User-Installed Serial Interface Components

The iRCB 44/10 board also provides you with several clocking options for the serial interface. You can install oscillators into sockets Y1 and Y2. Y1 can provide either MCLK or SCLK; Y2 can provide only SCLK. You can determine the SCLK frequency with the following formulas:

$$\text{Y1 SCLK} = \frac{\text{Oscillator Frequency}}{4} \qquad \text{Y2 SCLK} = \frac{\text{Oscillator Frequency}}{2}$$

The ALE signal can also be used for the serial clock. The frequency of the ALE signal is approximately 2MHz. By installing jumper E66-E67, you route the ALE signal through the flip-flop, divide-by-two, and produce an SCLK of approximately 1MHz. Refer to Chapter 11 for more details of the ALE option.

13.5.2.5 Repeater Interface

The iRCB 44/10 board supports an on-board repeater with the addition of 75174 and 75175 (RS485 transmitter and receiver) components. These components are not shipped with the board. In order to use the on-board repeater option, must install the 75174 and the 75175 into the sockets provided on the board. This repeater can be configured to repeat from the P1 connector to the J2 connector or from the J2 connector to the P1 connector. The location of the master node determines how the repeater should be configured. Also, the RTS signal lines require biasing resistors. Figure 13-4 shows the DCM core repeater configuration and the biasing arrangement for these signal lines. The biasing resistors should be 470 ohms. Table 13-9 specifies where the resistors and RS485 components must be installed for the repeater. Refer to Figure 13-10 for the location of these sockets on the board.

Table 13-9. Repeater Sockets/Components

Socket	Device	Biasing Resistors	Comments
U3 U4	75174 75175	R11 and R16	Master on P1 connector Repeats from P1 to J2
U9 U10	75174 75175	R18 and R22	Master on J2 connector Repeats from J2 to P1

When a repeater is present, both BITBUS segments must be terminated at both ends. The resistors' type and value are the same as those described for the serial interface. Table 13-10 lists the socket locations for the termination resistors. Refer to Figure 13-10 for the location of these sockets on the board.

Table 13-10. Termination Sockets

Socket	Function
R13 R14	Repeater RTS signals termination Repeater DATA signals termination

13.5.2.6 Interrupt Sources

The iRCB 44/10 board supports interrupt sources from the iSBX interface or from the parallel port. The iSBX I/O Expansion bus interface connection is hardwired corresponding to the iSBX 344 Multimodule board parallel interface requirements. When the iSBX interface is not using the parallel interface, you can install jumpers to connect an interrupt from the parallel interface as specified in Table 13-11.

Table 13-11. Interrupt Sources Jumper Options

Jumpers	Function
E40 - E41	MINT1 to INT1*
E41 - E42	RGND/EXT INT to INT1*

13.5.2.7 Numerical List Of Jumpers

This section provides an overview of the jumpers on the iRCB 44/10 board. Figure 13-11 shows the approximate location of each jumper on the iRCB 44/10 board. Table 13-12 lists all the jumpers and their functions, and Table 13-13 lists all of the jumpers that are installed in the default configuration of the board.

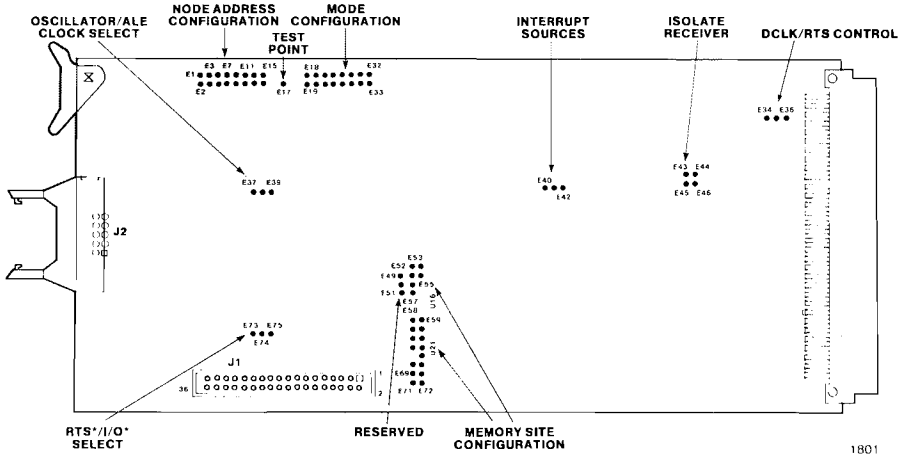


Figure 13-11. Jumper Location Diagram

Table 13-12. Numerical List Of Jumpers And Their Functions

Jumper Number	Functions
E1-E2	Sets bit 7 of node address register
E3-E4	Sets bit 6 of node address register
E5-E6	Sets bit 5 of node address register
E7-E8	Sets bit 4 of node address register
E9-E10	Sets bit 3 of node address register
E11-E12	Sets bit 2 of node address register
E13-E14	Sets bit 1 of node address register
E15-E16	Sets bit 0 of node address register
E17	Test point
E18-E19	Reserved
E20-E21	Reserved
E22-E23	Indicate presence of byte FIFO
E24-E25	Memory addressing configuration
E26-E27	EA to 8044
E28-E29	Reserved
E30-E31 and E32-E33	Set BITBUS mode and bit rate
E34-E35§	Connects SCLK to processor
E35-E36	Ties RTS low for RTS output
E37-E38§	Selects oscillator as clock source
E38-E39	Selects ALE as clock source
E40-E41	MINT to INT 1
E41-E42	EXT INT to INT 1
E43-E44§	Removes SCLK receiver load for self-clocked mode
E45-E46§	Removes SCLK receiver load for self-clocked mode
E49-E50	Not used
E50-E51	Not used
E52 thru E57	Data site memory configuration E53-E55§ E56-E57§
E58 thru E72	Code site memory configuration E58-E59§ E60-E62§ E63-E65§ E71-E72§
E73-E74	Selects RTS* for transceiver direction (synch. mode)
E74-E75§	Selects I/O* for transceiver direction (self-clocked mode)

Table 13-13. Installed Jumpers (As-Shipped Configuration)

Jumper Connections		
E34-E35§ E45-E46§ E58-E59§ E71-E72§	E37-E38§ E53-E55§ E60-E62§ E74-E75§	E43-E44§ E56-E57§ E63-E65§

13.6 PROGRAMMING CONSIDERATIONS

The iRCB 44/10 has two programming interfaces: the DCM Controller and the user I/O.

13.6.1 DCM CONTROLLER PROGRAMMING

Programming the DCM component is optional. The firmware provided already handles communications and I/O functions. If you desire additional capabilities, you can add user tasks. Refer to Chapters 5 and 6 for more information on creating user tasks. You must write these tasks in either ASM 51 or PL/M 51 and stored in a memory device to be located in the Universal Memory site.

When adding user tasks to the iRCB 44/10 board, it is usually not necessary to know any I/O port addresses because the firmware handles both the parallel and serial interfaces. For special cases where a user task needs to access an I/O port function directly, Table 13-14 lists the addresses. These addresses are either memory-mapped I/O or bit addressed I/O as indicated in the table.

The iRCB 44/10 board can be used as an intelligent BITBUS repeater. In this application, a user task can perform the following functions through an iRCB 44/10 board and an iSBX 344 board:

- Transfer a faster BITBUS system to a slower BITBUS system
- Transfer a slower BITBUS system to a faster BITBUS system
- Extend the number of BITBUS nodes at the same speed

This is useful for connecting a group of local nodes to some distant node which requires a lower transmission speed. The iRCB 44/10 board is designed to accommodate this by appropriately connecting to the iSBX signals for byte FIFO operation.

Table 13-14. DCM Controller (8044) I/O Addressing On iRCB 44/10 Board

Function	Address	Read	Write	Bit
Port A	FFC0H	X	X	
Port B	FFC1H	X		
Port C	FFC2H	X	X	
MCS0	FF80H-FF87H FF00,FF01 (1)	X	X	
MCS1	FF88H-FF8FH	X	X	
Red LED	90H	X	X	X
Green LED	91H	X	X	X
RDY/NE*	B4H	X	X	X
Node Address	FFFFH	X		
Configuration	FFFEH	X		
OPT0 (2)	92H	X	X	X
OPT1	93H	X	X	X
INT0 (2)	B2H	X		X
INT1 (2)	B3H	X		X

Notes: 1. For DCM FIFO operation (Byte FIFO present)
2. DCM FIFO signals when byte FIFO is present

13.6.2 I/O PROGRAMMING

The iRCB 44/10 board provides parallel I/O and iSBX I/O Expansion Bus connectors. These I/O functions can be programmed locally or remotely, using the RAC function described in Chapter 8 or locally by a user task as described in the following sections.

13.6.2.1 PARALLEL I/O

The iRCB 44/10 board can be programmed locally by a user task by either using the RAC function or by directly reading and writing to the I/O ports. Details of the RAC function are left to Chapter 8 . Direct operations are classified into four groups: input, output, mixed, or bit operations. Note that for all operations, the logical state of the I/O pins are inverted.

Input operations are performed by simply reading the desired port. On reset all ports are configured as inputs. If a port has been used as an output, writing zeros to it reprograms it to an input.

Output operations are performed by simply writing to the desired port. On power-up the 1k ohm pull-up resistors provide all output pins with a logic high state. When a one is written to the port latch, an open collector driver provides a logic low state.

Mixed operations involve mixing input and output on a single port. The iRCB 44/10 port structure is designed to support this capability; however, you must be careful to always write a zero to input bits when writing to output bits. ANDing the output data with a mask accomplishes this operation. The mask consists of ones in output bit positions and zeros in input bit positions.

Bit operations may be added to any of the operations above. The capability to read back a port allows simple AND, OR, and XOR operations to perform bit set, reset, and toggle, respectively. When performing bit operations on mixed ports, it is important to use the previously mentioned mask function. Figure 13-12 provides an example of a bit set and mask operation. The RAC function supports the bit set, reset, and toggle capabilities both locally and remotely.

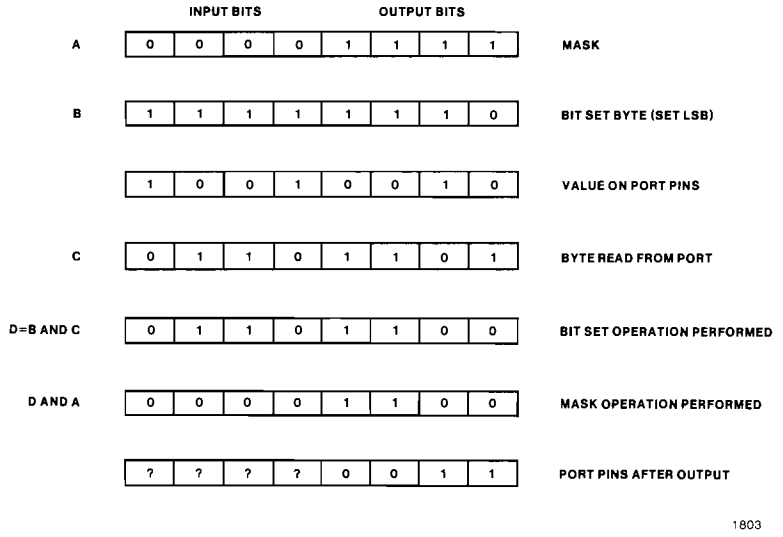


Figure 13-12. Example Of Bit Set And Mask Operation

13.6.2.2 iSBX I/O EXPANSION PROGRAMMING

Programming this interface depends upon the function of the expansion module. Refer to the hardware reference manual of the iSBX board for more information.

***

CHAPTER 14
SERVICE INFORMATION

14.1 INTRODUCTION

This chapter provides the required service diagrams, plus service and repair assistance instructions for the iSBX 344 BITBUS Controller MULTIMODULE board and the iRCB 44/10 Remote Controller Board.

14.2 SERVICE DIAGRAMS

The parts location diagram and schematic diagram for the iSBX 344 BITBUS Controller MULTIMODULE board are shown in Figures 14-1 and 14-2, respectively.

The parts location diagram and schematic diagram for the iRCB 44/10 Remote Controller Board are shown in Figures 14-3 and 14-4, respectively.

On the schematic diagrams, a signal mnemonic that ends with an asterisk (for example, ALE*) is active low. Conversely, a signal mnemonic without an asterisk (for example, ALE) is active high.

14.3 SERVICE AND REPAIR ASSISTANCE

United States customers can obtain service and repair assistance by contacting the Intel Product Service Marketing Administration in Phoenix, Arizona. Customers outside the United States should contact their sales source (Intel Sales Office or Authorized Distributor) for service information and repair assistance.

Before calling the Product Service Marketing Administration, you should have the following information available:

a. The date on which you received the product.

b. The complete model number (including the dash number) and serial number for the product. These numbers are stamped onto Intel printed circuit boards.

c. Your shipping and billing addresses.

d. A purchase order number, for billing purposes if your Intel product warranty has expired.

e. Any applicable extended warranty agreement information.

Use the following numbers for contacting the Intel Product Service Marketing Administration group:

Regional Telephone Numbers:

Western Region:	602-869-4862
Midwestern Region:	602-869-4392
Eastern Region:	602-869-4045
International:	602-869-4391

TWX Numbers:

910 - 951 - 1330
910 - 951 - 0687

Always contact the Product Service Marketing Administration group before returning a product to Intel for repair. You will be given a repair authorization number, shipping instructions, and other important information that will help Intel provide you with fast, efficient service. If you are returning the product because of damage sustained during shipment, or if the product is out of warranty, a purchase order is required before Intel can initiate the repair.

In preparing the product for shipment to Intel, use the original factory packing material, if possible. If this material is not available, wrap the product in cushioning material such as Air Cap TH-240, manufactured by the Sealed Air Corporation, Hawthorne, N. J. Then enclose in a heavy duty corrugated shipping carton, and label "FRAGILE" to ensure careful handling. Ship it only to the address specified by the Intel Product Service Marketing Administration personnel.

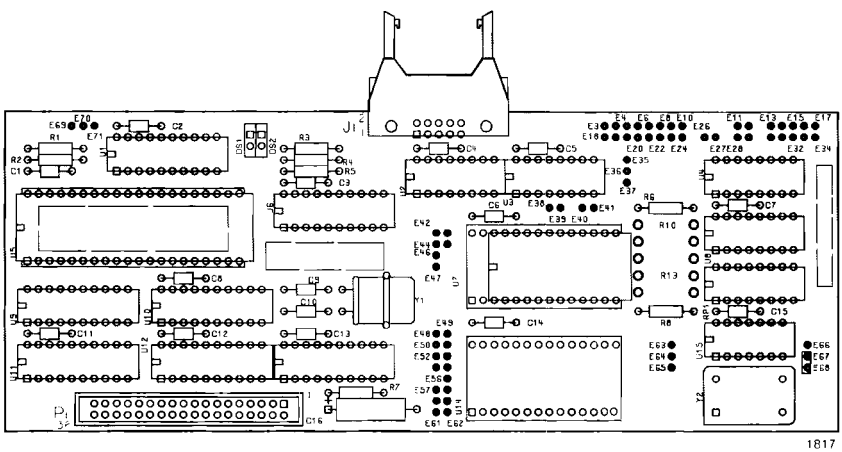
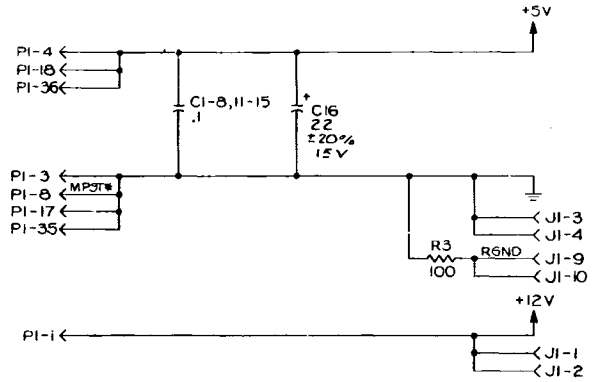


Figure 14-1. 1SBX™ 344 BITBUS™ Controller MULTIMODULE™
Parts Location Diagram



DEVICE TABLE

REFERENCE DESIGNATIONS	DEVICE TYPE	POWER PINS GND	+5V	+5VB	+12V	UNUSED LOGIC ELEMENT OUTPUT PINS
U5	8044	20	40			
[4] U12	PAL	10	20			
[4] U10	PAL	10	20			
[4] U1	PAL	10	20			
U15	74LS74	7	14			
U3	75174	8	16			
U2	75175	8	16			
U13	74LS240	10	20			U13-12
U4,8	74LS258	8	16			
U6	74LS373	10	20			
U9,11	74LS374	10	20			

NOTES; UNLESS OTHERWISE SPECIFIED:

1. CAPACITANCE VALUES ARE MICROFARADS, +80%, -20%, 50V.
2. RESISTANCE VALUES ARE IN OHMS, ¼W, 5%.
3. AN ASTERISK FOLLOWING A SIGNAL NAME DENOTES NEGATION.

[4] THESE ARE PROGRAMMED PARTS.

[5] R10,13, Y2, AND U14 ARE CUSTOMER SUPPLIED AND INSTALLED

REF DESIGNATIONS		
LAST USED		NOT USED
U15	E71	R9,12,14
R13	Y2	
RP1	P1	
C16	J1	
DS2		

REVISIONS							
REV	DESCRIPTION	DFT	DATE	CHK	DATE	APVD	DATE
A	ECO 40-4029	DRN	[illegible]/83	MA	11/16/83	JE	11/23

DRN BY R. Tong	7/1/83	intel 3065 BOWERS AVE SANTA CLARA CALIF. 95051
CHK BY JGB	7/16/83	TITLE SCHEMATIC DIAGRAM iSBX 344
APVD [illegible]	7/21/83	
APVD		SIZE D | SCL C | CODE 40 | 0100 | DOCUMENT NUMBER 146019 | REV A
APVD		SCALE NONE | SHEET 1 OF 3

Figure 14-2. iSBX™ 344 BITBUS™ Controller MULTIMODULE™ Schematic Diagram (Sheet 1 of 3)

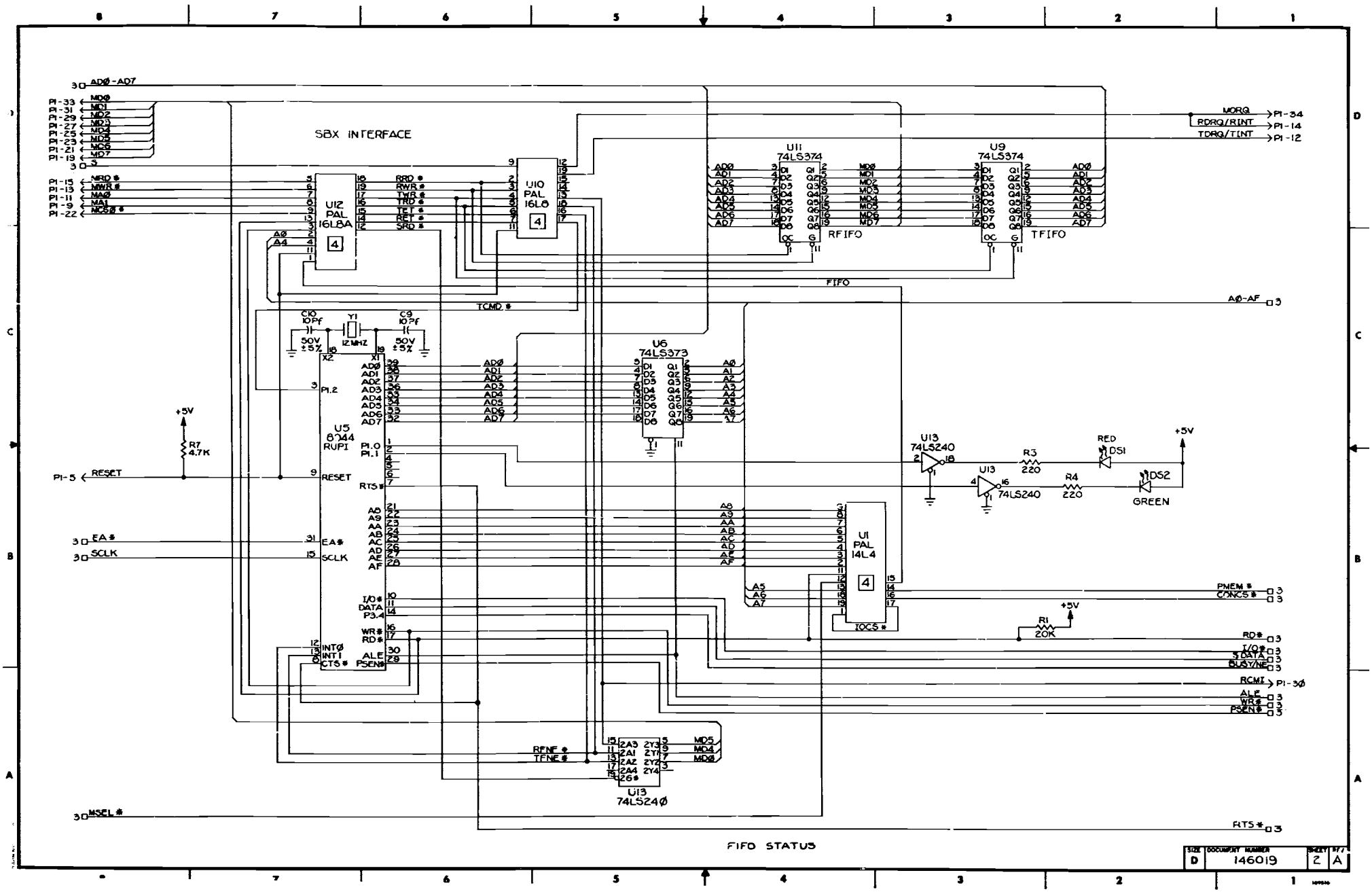


Figure 14-2. iSBX™ 344 BITBUS™ Controller MULTIMODULE™ Schematic Diagram
(Sheet 2 of 3)

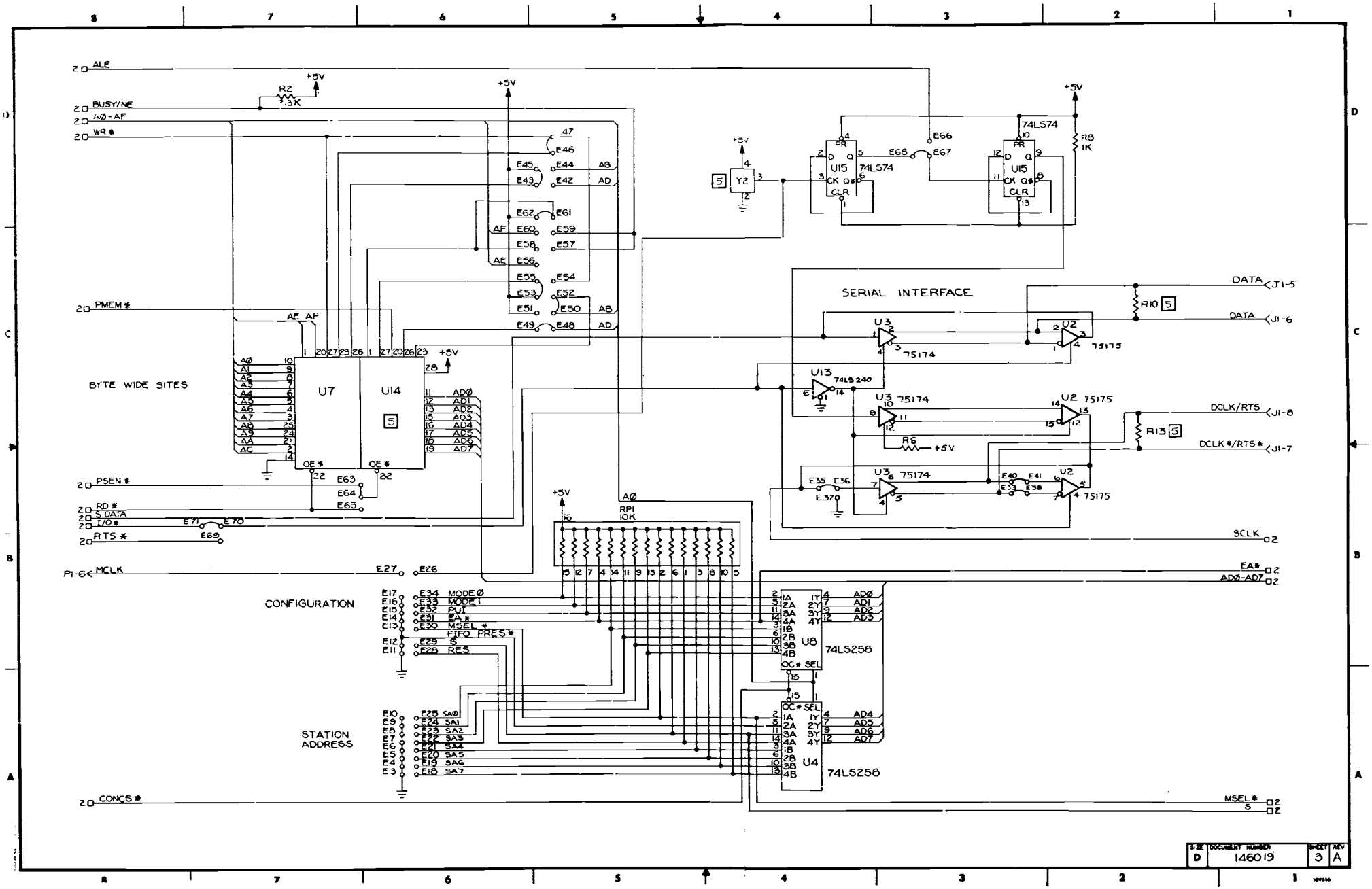


Figure 14-2. iSBX™ 344 BITBUS™ Controller MULTIMODULE™ Schematic Diagram
(Sheet 3 of 3)

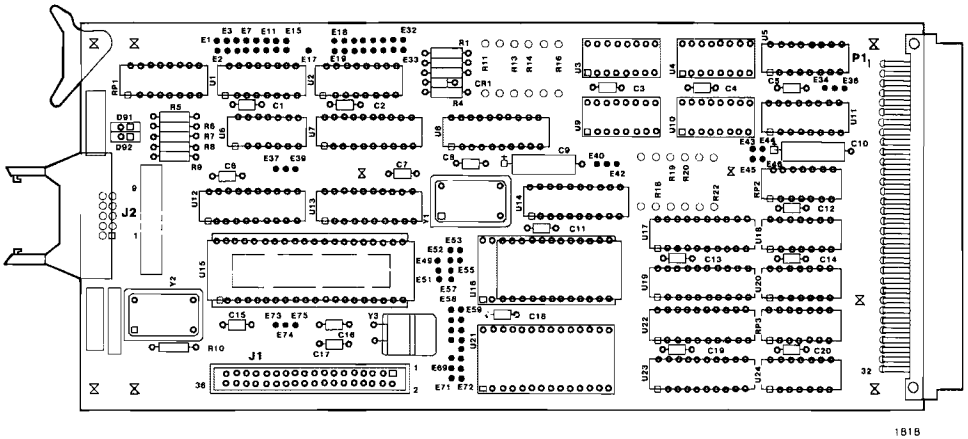
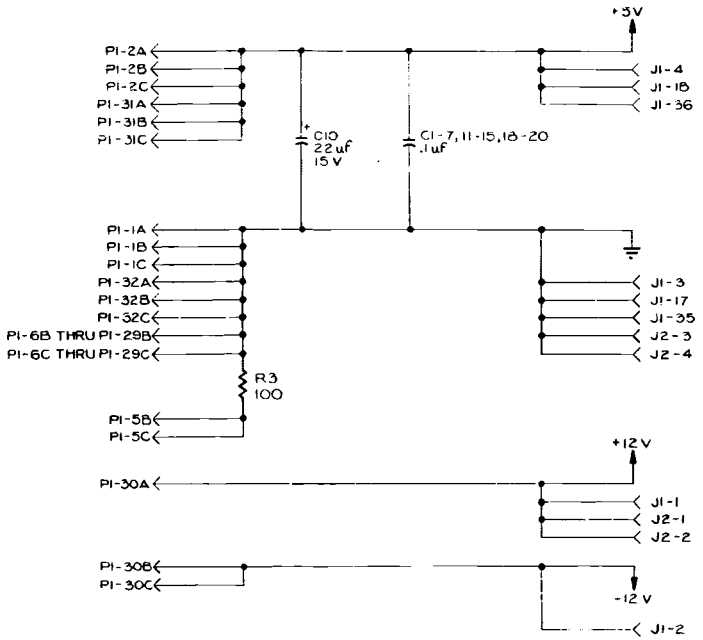


Figure 14-3. iRCB 44/10 Remote Controller Board Parts Location Diagram



DEVICE TABLE

REFERENCE DESIGNATIONS	DEVICE TYPE	POWER PINS GND	+5V	+12V	-12V	UNUSED LOGIC ELEMENT OUTPUT PINS
U15	8744	20	40			
U13	PAL14L4	10	20			
U7	PAL16L8A	10	20			
U18,20,24	7406	7	14			
U6	74LS74	7	14			
U11	75174	8	16			
U5	75175	8	16			
U8,17,19,22	74LS240	10	20			
U1,2	74LS256	8	16			
U14,23	74LS273	10	20			
U12	74LS373	10	20			
U16	MEM	14	28			
[4] U21	MEM	14	28			
[4] U3,9	75174	8	16			
[4] U4,10	75,74	8	16			

NOTES; UNLESS OTHERWISE SPECIFIED:

1. CAPACITANCE VALUES ARE IN MICROFARADS, +80%, -20%, 50V.
2. RESISTANCE VALUES ARE IN OHMS, ¼ W, 5%.
3. AN ASTERISK FOLLOWING A SIGNAL NAME DENOTES NEGATION.

[4] CUSTOMER SUPPLIED AND INSTALLED PART.

[5] THESE ARE PROGRAMMED PARTS.

REF DESIGNATIONS	
LAST USED	NOT USED
U24 RP3 C20 J52 P1 J2 E75 Y3 R22	E47,48 R12,15,17,21

REVISIONS

REV	DESCRIPTION	DFT	DATE	CHK	DATE	APVD	DATE
A	ECO 40-4098			KA	11/1/83	JE	3/1/84

UNLESS OTHERWISE SPECIFIED:
1. DIMENSIONS ARE IN INCHES
2. BREAK ALL SHARP EDGES.
3. DO NOT SCALE DRAWING.
4. TOLERANCES: ANGLES ±, XX ±, XXX ±

QUANTITY PER DASH NO.

ITEM NO. | DESCRIPTION | PARTS LIST

SIGNATURE	DATE
DRN BY L. Trong	9/1/83
CHK BY Greg Ick	9/1/83
ENGR APVD Jeff Lawrence	9/1/83
APVD	
APVD	

intel 3065 BOWERS AVE. SANTA CLARA CALIF. 95054

TITLE: SCHEMATIC DIAGRAM iRCB 44/10

SIZE	DCL	CL CODE	RLSE LVL	DOCUMENT NUMBER	REV
D	40	0100		146081	A

SCALE NONE | SHEET 1 OF 5

Figure 14-4. iRCB 44/10 Remote Controller Board Schematic Diagram (Sheet 1 of 5)

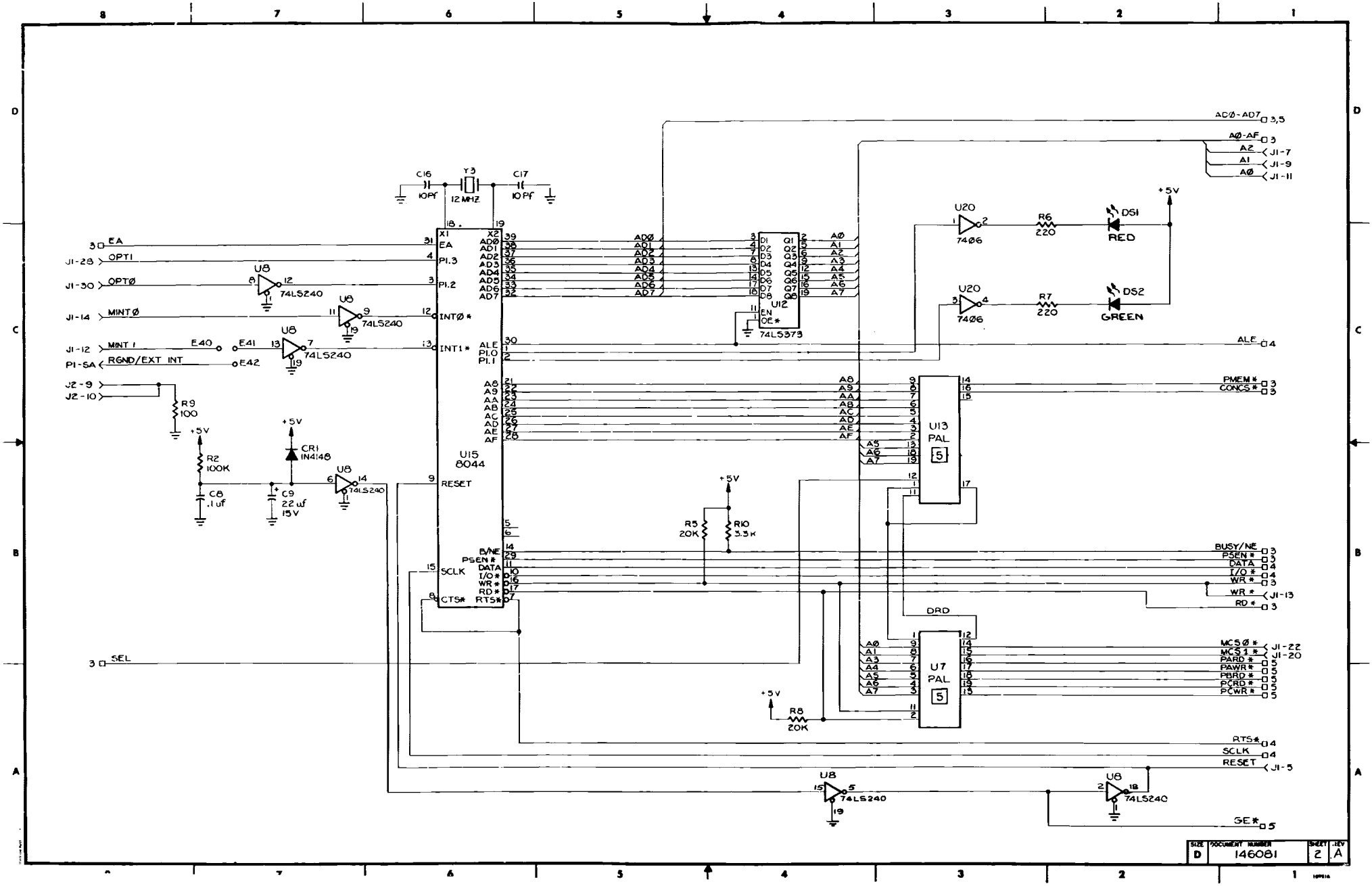


Figure 14-4. iRCB 44/10 Remote Controller Board Schematic Diagram
(Sheet 2 of 5)

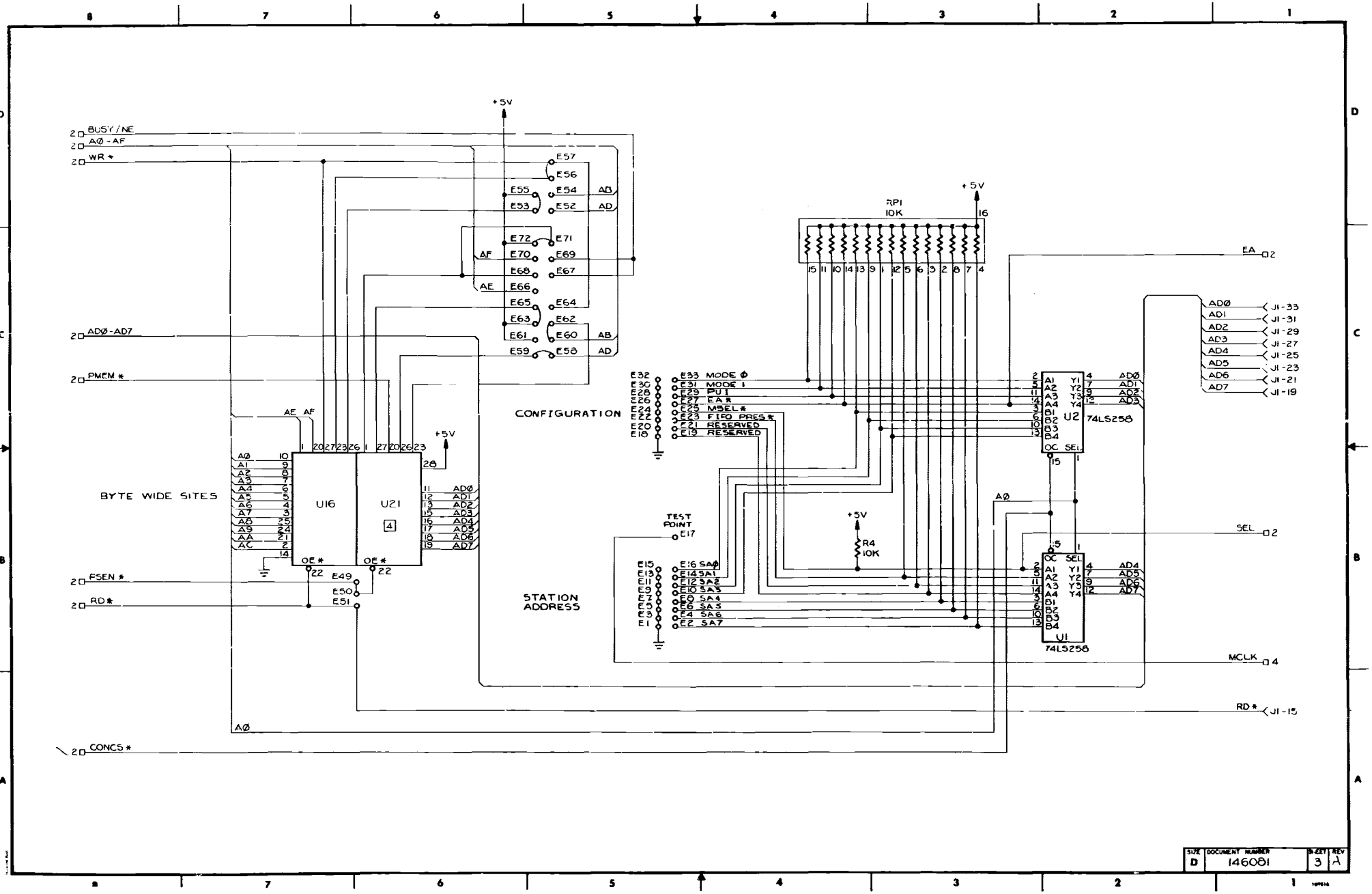


Figure 14-4. iRCB 44/10 Remote Controller Board Schematic Diagram
(Sheet 3 of 5)

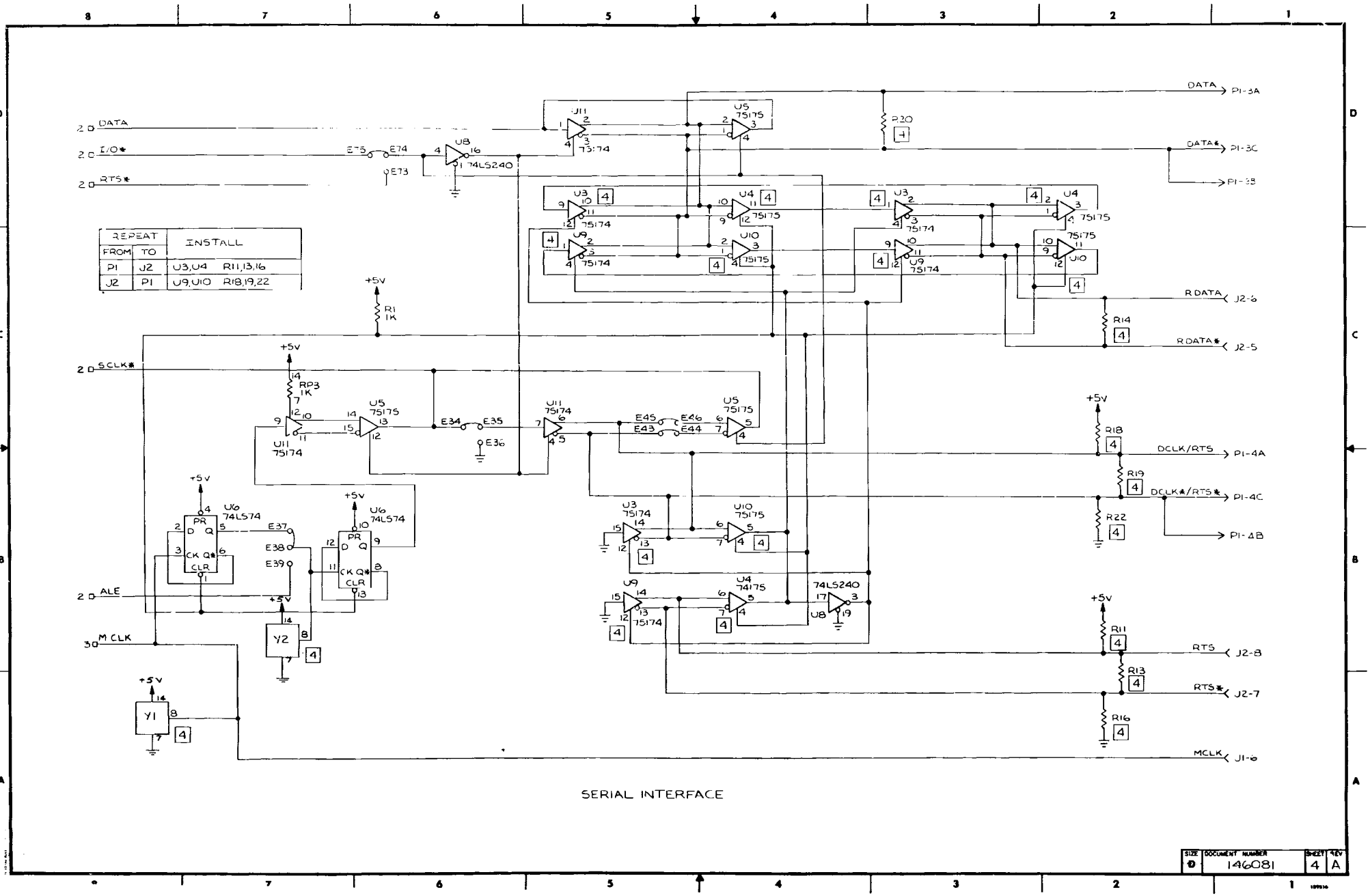


Figure 14-4. iRCB 44/10 Remote Controller Board Schematic Diagram
(Sheet 4 of 5)

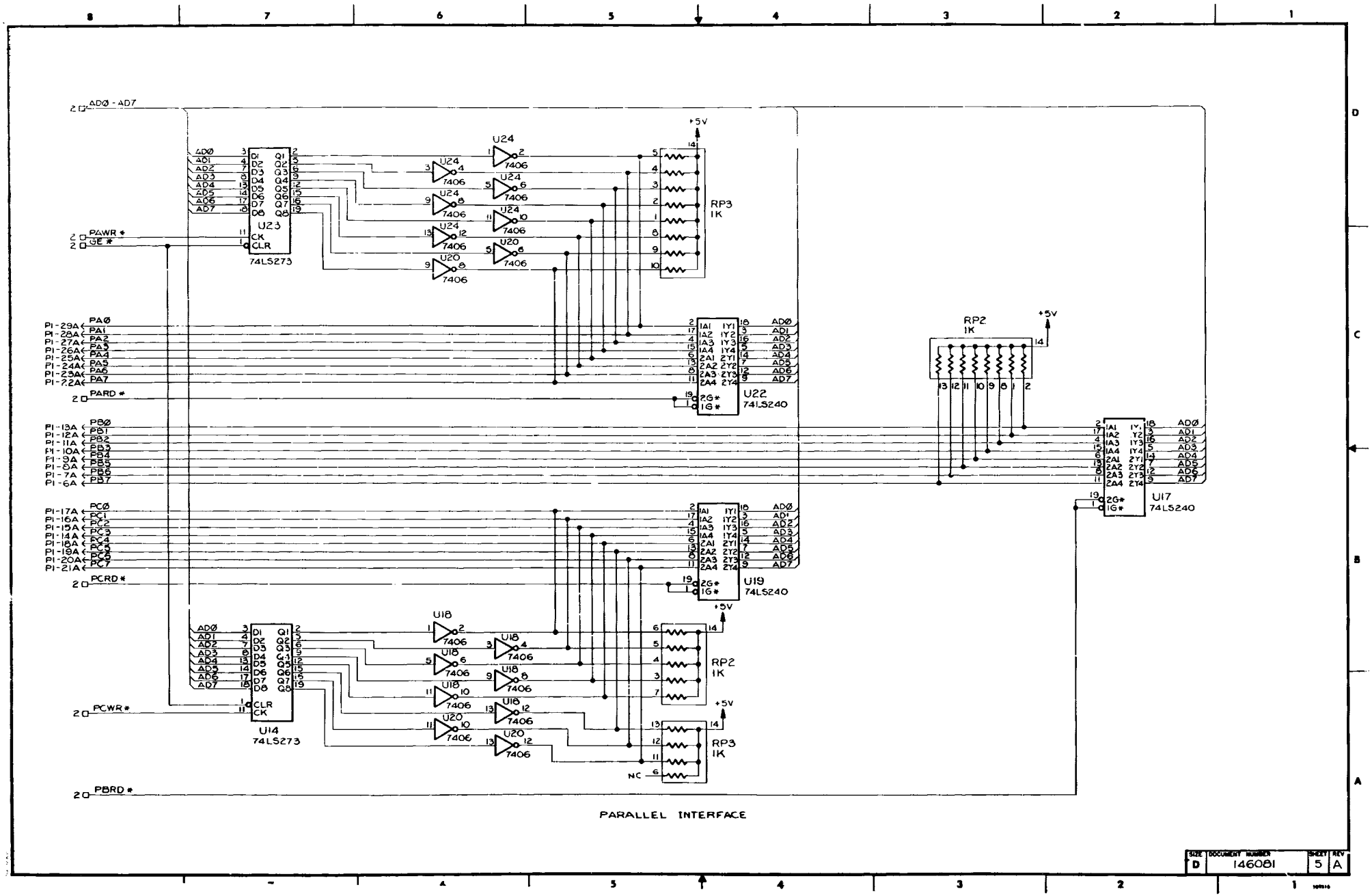


Figure 14-4. iRCB 44/10 Remote Controller Board Schematic Diagram
(Sheet 5 of 5)

APPENDIX A
IMPORTANT DCM AND iRMX™ 51 FILES

This appendix lists the contents of some DCM and iRMX 51 files that you need to use in configuring either your DCM system or your stand-alone iRMX 51 system. This list contains the names of the files and their functions:

- DCM44A.LIT — The system and RAC command/response literal declarations for ASM51 programs.
- DCM44P.LIT — The system and RAC command/response literal declarations for PL/M 51 programs.
- RMX51A.LIT — The iRMX 51 literal declarations for ASM51 tasks.
- RMX51P.LIT — The iRMX 51 literal declarations for PL/M 51 tasks.
- RMX51A.EXT — The iRMX 51 external declarations for ASM51 tasks.
- RMX51P.EXT — The iRMX 51 external declarations for PL/M 51 tasks.
- RMX51A.MAC — The file which contains the ITD macro. You can use this macro in your iRMX 51 applications.
- RMX51A.CFG — The configuration file for the iRMX 51 Executive.

The following sections list the contents of these files. Refer to Chapter 7 for more information about configuring your system.

A.1 CONTENTS OF DCM44A.LIT

Figure A-1 lists the contents of the INCLUDE file, DCM44A.LIT.

```
;/*************************************************************/
;/*
;/* TITLE:           iDCM 44 COMMUNICATIONS FIRMWARE: LITERAL
;/*                  DECLARATIONS
;*
;/* RELEASE:         1.0
;/*
;/* DESCRIPTION:     THIS MODULE CONTAINS THE LITERAL DECLARATIONS
;/*                  FORDCM AND RAC COMMANDS AND RESPONSES AND
;/*                  CONFIGURATION PARAMETERS.
;/*
;/*************************************************************/

; Message Status constants

    E_SERVICE_OK             EQU     00H
    E_PROTOCOL_ERROR         EQU     91H
    E_NO_DESTINATION_DEVICE  EQU     93H
    E_ACCESS_PROTECTED       EQU     94H
    E_RAC_PROTECTED          EQU     95H
    E_UNKNOWN_RAC_COMMAND    EQU     96H

; RAC message constants

    RESET_STATION_MSK        EQU     00H
    CREATE_TASK_MSK          EQU     01H
    DELETE_TASK_MSK          EQU     02H
    GET_FUNCTION_IDS_MSK     EQU     03H
    RAC_PROTECT_MSK          EQU     04H
    EXT_IO_READ_MSK          EQU     05H
    EXT_IO_WRITE_MSK         EQU     06H
    EXT_IO_UPDATE_MSK        EQU     07H
    EXT_UPLOAD_MSK           EQU     08H
    EXT_DNLOAD_MSK           EQU     09H
    EXT_IO_OR_MSK            EQU     0AH
    EXT_IO_AND_MSK           EQU     0BH
    EXT_IO_XOR_MSK           EQU     0CH
    INT_WRITE_MSK            EQU     0DH
    INT_READ_MSK             EQU     0EH

;User Task Location Constants

    MIN_USER_INT_DATA_BIT    EQU     048H
    MAX_USER_INT_DATA_BIT    EQU     07FH
```

Figure A-1. Contents of DCM44A.LIT

```
MIN_USER_INT_DATA_ADR   EQU     029H
MAX_USER_INT_DATA_ADR   EQU     02FH

MIN_USER_EXT_DATA_ADR   EQU     00300H
MAX_USER_EXT_DATA_ADR   EQU     0EFFFH

MIN_USER_CODE_ADR       EQU     01000H
MAX_USER_CODE_ADR       EQU     0FFEFH

FIRST_USER_ITD_ADR      EQU     0FFF0H
```

Figure A-1. Contents of DCM44A.LIT (continued)

A.2 CONTENTS OF DCM44P.LIT

Figure A-2 lists the contents of the INCLUDE file DCM44P. LIT.

```
/*****************************************************************/
/*
/*  TITLE:          iDCM 44 COMMUNICATIONS FIRMWARE: LITERAL
/*                  DECLARATIONS FOR PL/M 51 CODE
/*
/*  RELEASE:        1.0
/*
/*  DESCRIPTION:    THIS MODULE CONTAINS THE LITERAL DECLARATIONS FOR
/*                  DCM AND RAC COMMANDS AND RESPONSES AND
/*                  CONFIGURATION PARAMETERS.
/*
/*****************************************************************/

/* Message Status constants */

    DECLARE E_SERVICE_OK            LITERALLY   '00H';
    DECLARE E_PROTOCOL_ERROR        LITERALLY   '91H';
    DECLARE E_NO_DESTINATION_DEVICE LITERALLY   '93H';
    DECLARE E_ACCESS_PROTECTED      LITERALLY   '94H';
    DECLARE E_RAC_PROTECTED         LITERALLY   '95H';
    DECLARE E_UNKNOWN_RAC_COMMAND   LITERALLY   '96H';

/* RAC message constants */

    DECLARE RESET_STATION_MSK       LITERALLY   '00H';
    DECLARE CREATE_TASK_MSK         LITERALLY   '01H';
    DECLARE DELETE_TASK_MSK         LITERALLY   '02H';
    DECLARE GET_FUNCTION_IDS_MSK    LITERALLY   '03H';
```

Figure A-2. Contents of DCM44P.LIT

```
    DECLARE RAC_PROTECT_MSK          LITERALLY   '04H';
    DECLARE EXT_IO_READ_MSK          LITERALLY   '05H';
    DECLARE EXT_IO_WRITE_MSK         LITERALLY   '06H';
    DECLARE EXT_IO_UPDATE_MSK        LITERALLY   '07H';
    DECLARE EXT_UPLOAD_MSK           LITERALLY   '08H';
    DECLARE EXT_DNLOAD_MSK           LITERALLY   '09H';
    DECLARE EXT_IO_OR_MSK            LITERALLY   '0AH';
    DECLARE EXT_IO_AND_MSK           LITERALLY   '0BH';
    DECLARE EXT_IO_XOR_MSK           LITERALLY   '0CH';
    DECLARE INT_WRITE_MSK            LITERALLY   '0DH';
    DECLARE INT_READ_MSK             LITERALLY   '0EH';

/* User Task Location Constants */

    DECLARE MIN_USER_INT_DATA_BIT    LITERALLY   '048H';
    DECLARE MAX_USER_INT_DATA_BIT    LITERALLY   '07FH';

    DECLARE MIN_USER_INT_DATA_ADR    LITERALLY   '029H';
    DECLARE MAX_USER_INT_DATA_ADR    LITERALLY   '02FH';

    DECLARE MIN_USER_EXT_DATA_ADR    LITERALLY   '00300H';
    DECLARE MAX_USER_EXT_DATA_ADR    LITERALLY   '0EFFFH';

    DECLARE MIN_USER_CODE_ADR        LITERALLY   '01000H';
    DECLARE MAX_USER_CODE_ADR        LITERALLY   '0FFEFH';

    DECLARE FIRST_USER_ITD_ADR       LITERALLY   '0FFF0H';
```

Figure A-2. Contents of DCM44P.LIT (continued)

A.3 CONTENTS OF RMX51A.LIT

Figure A-3 lists the contents of the INCLUDE file, RMX51A.LIT.

```
;
;   iRMX 51 Status literals
;

E_OK          EQU     00H
E_EXIST       EQU     80H
E_MAX_TASKS   EQU     81H
E_REGS        EQU     82H
E_FID         EQU     83H
E_BUFFERS     EQU     84H
```

Figure A-3. Contents of RMX51A. LIT

A.4 CONTENTS OF RMX51P.LIT

Figure A-4 lists the contents of the INCLUDE file, RMX51P.LIT.

```
/*
*   iRMX 51 Status literals
*/

E$OK            LITERALLY       '00H'
E$EXIST         LITERALLY       '80H'
E$MAX$TASKS     LITERALLY       '81H'
E$REGS          LITERALLY       '82H'
E$FID           LITERALLY       '83H'
E$BUFFERS       LITERALLY       '84H'
```

Figure A-4. Contents of RMX51P.LIT

A.5 CONTENTS OF RMX51A.EXT

Figure A-5 lists the contents of the INCLUDE file, RMX51A.EXT.

```
EXTRN CODE  ( RQCREATETASK, RQDELETETASK, RQGETFUNCTIONIDS,
              RQDISABLEINTERRUTPT )
EXTRN CODE  ( RQENABLEINTERRUPT, RQSENDMESSAGE, RQSETINTERVAL,
              RQWAIT )
EXTRN CODE  ( RQALLOCATE, RQDEALLOCATE )

EXTRN DATA  ( RQTASKID, RQTASKPRIORITY, RQCLOCKUNIT )

EXTRN DATA  ( RQINITSTATUS )
```

Figure A-5. Contents of RMX51A.EXT

A.6 CONTENTS OF RMX51P.EXT

Figure A-6 lists the contents of the INCLUDE file, RMX51P.EXT.

```
/*
*   iRMX 51 system call external procedure declarations for PLM/51
*/

RQCREATETASK:  PROCEDURE BYTE EXTERNAL:
    END RQCREATETASK;
RQDELETETASK:  PROCEDURE BYTE EXTERNAL;
    END RQDELETETASK;
RQGETFUNCTIONIDS:  PROCEDURE EXTERNAL;
    END RQGETFUNCTIONIDS;
RQDISABLEINTERRUPT:  PROCEDURE EXTERNAL;
    END RQDISABLEINTERRUPT;
RQENABLEINTERRUPT:  PROCEDURE EXTERNAL;
    END RQENABLEINTERRUPT;
RQSENDMESSAGE:  PROCEDURE EXTERNAL;
    END RQSENDMESSAGE;
RQSETINTERVAL:  PROCEDURE EXTERNAL;
    END RQSETINTERVAL;
RQWAIT:  PROCEDURE WORD EXTERNAL;
    END RQWAIT;
RQALLOCATE:  PROCEDURE BYTE EXTERNAL;
    END RQALLOCATE;
RQDEALLOCATE:  PROCEDURE EXTERNAL;
    END RQDEALLOCATE;

/*
*   iRMX 51 system variable external declarations
*/

DECLARE
    RQTASKID        BYTE   EXTERNAL   MAIN,
    RQCLOCKUNIT     WORD   EXTERNAL   MAIN,
    RQINITSTATUS    BYTE   EXTERNAL   AUXILIARY;
```

Figure A-6. Contents of RMX51P.EXT

A.7 CONTENTS OF RMX51A.MAC

Figure A-7 lists the contents of the macro file, RMX51A.MAC.

```
%DEFINE(ITD( INITIALPC, STACKLENGTH, FUNCTIONID, REGISTERBANKSELECT,
             PRIORITY, INTERRUPTVECTOR, NEXTITD )) LOCAL ITDSTART

(
%
%' Initial Task Descriptor Macro
%
%ITDSTART:
            DW          1010101001010101B
            DW          %ITDSTART - %INITIALPC
            DB          %STACKLENGTH
            DB          %FUNCTIONID
            DB          EVAL( (%REGISTERBANKSELECT * 16) + %PRIORITY)
            DB          0
            DB          %INTERRUPTVECTOR
%IF (%EQS(%NEXTITD,0FFFFH)) THEN
(
            DW          0FFFFH
)
ELSE
)
            DW          %ITDSTART - %NEXTITD
)
(
FI
)
```

Figure A-7. Contents of RMX51A.MAC

A.8 CONTENTS OF RMX51A.CFG

Figure A-8 lists the contents of the configuration file, RMX51A.CFG.

```
;
;          iRMX™ 51 Configuration File
;

 PUBLIC  RQFIRSTITD, RQCLOCKPRIORIYT, RQ$CLOCKTICK
 PUBLIC  RQPOOLTOP, RQSYSBUFSIZE

RQFIRSTITD        EQU     ITD1      ;address of first task descriptor
RQCLOCKPRIORITY   EQU     5         ;priority of system clock
RQCLOCKTICK       EQU     -1000     ;(negative of) clock cycles/tick
RQPOOLTOP         EQU     191       ;address of top of free space pool
RQSYSBUFSIZE      EQU     20        ;free space buffer size
```

Figure A-8. Contents Of RMX51A.CFG

***

RED page numbers indicate the primary references; underscored page numbers indicate figure or table references.

***